A NEW MONTE-CARLO METHOD FOR ESTIMATING THE FAILURE PROBABILITY OF AN n-COMPONENT SYSTEM

Richard M. Karp† and Michael G. Luby†

Computer Science Division

University of California

Berkeley, CA 94720

ABSTRACT

A new formula for the probability of a union of events is used to express the failure probability of an n-component system. A very simple Monte-Carlo algorithm based on the new probability formula is presented. The input to the algorithm gives the failure probabilities of the n components of the system and a list of the failure sets of the system. The output is an unbiased estimator of the failure probability of the system. We show that the average value of the estimator over many runs of the algorithm tends to converge quickly to the failure probability of the system. The overall time to estimate the failure probability with high accuracy compares very favorably with the execution times of other methods used for solving this problem.

A NEW MONTE-CARLO METHOD FOR ESTIMATING THE FAILURE PROBABILITY OF AN n-COMPONENT SYSTEM

Richard M. Karp† and Michael G. Luby†

Computer Science Division

University of California

Berkeley, CA 94720

1. The Reliability Problem

We assume throughout this paper that an instance of the n-component reliability problem is specified by the following data:

(a) for each component i, where $1 \leq i \leq n$, a failure probability p_i. Component i is failing with probability p_i and working with probability $1 - p_i$ independently of all other components in the system. The assumption made here that components are s-independent is convenient, but not essential, for the development that follows.

(b) a specification of the combinations of component states which cause the overall system to fail. As we describe below, this specification consists of a list of m failure sets.

Given these data, the problem is to estimate the failure probability of the system.

In order to discuss how failure sets are specified, and how failure sets together with the failure probabilities of components determine the failure probability of the system, we require further definitions. A system state is an n-

† Research supported by NSF grant MCS-8105217.

tuple $(b_1, \ldots, b_n)$ where $b_i = 0$ if component i is failing, and $b_i = 1$ if component i is working. There are 2^n different system states and the probability of any particular system state $(b_1, \ldots, b_n)$ is $\prod_{i=1}^{n} p_i^{1-b_i} (1-p_i)^{b_i}$. For example, in an eight-component system, the system state (0, 1, 1, 0, 1, 0, 1, 1) has probability $p_1(1-p_2)(1-p_3)p_4(1-p_5)p_6(1-p_7)(1-p_8)$.

Let $(c_1, \ldots, c_n)$ be an n-tuple, each component of which is either 0, 1, or *. Such an n-tuple represents a set F of system states, according to the following rule: $(b_1, \ldots, b_n)$ is an element of F provided that $c_i = 0$ implies $b_i = 0$, $c_i = 1$ implies $b_i = 1$, and $c_i = *$ implies b_i may be either 0 or 1. Thus, the 8-tuple (0, *, 1, 1, 1, *, 0, 1) represents the following set of four system states: {(0, 0, 1, 1, 1, 0, 0, 1), (0, 0, 1, 1, 1, 1, 0, 1), (0, 1, 1, 1, 1, 0, 0, 1) and (0, 1, 1, 1, 1, 1, 0, 1)}. A set F represented in this way by an n-tuple $(c_1, \ldots, c_n)$ is called a failure set provided that each system state in F is a failure state of the system.

Let S be the set of all failure states of the system. We assume that S is specified as the union of failure sets $F_1, F_2, \ldots, F_m$, each of which is described by an n-tuple of 0's, 1's and *'s. Thus $Pr[S]$, which is the failure probability of the system, can be written as $Pr[\bigcup_{k=1}^{m} F_k]$.

The probability of a failure set F is the sum of the probabilities of the system states contained in F:

$$Pr[F] = \sum_{s \in F} Pr[s]. \tag{1.1}$$

Alternatively, if the n-tuple $(c_1, \ldots, c_n)$ specifies F, then

$$Pr[F] = \prod_{i=1}^{n} p_i^{c_i'} (1-p_i)^{c_i''} \quad \text{where}$$

$$c_i' = \begin{cases} 1 & \text{if } c_i = 0 \\ 0 & \text{if } c_i = 1 \text{ or } * \end{cases}$$

and

$$c_i'' = \begin{cases} 1 & \text{if } c_i = 1 \\ 0 & \text{if } c_i = 0 \text{ or } * \end{cases}$$

For example, $Pr[(0, *, 1, 1, 1, *, 0, 1)]$ is

$$p_1(1-p_3)(1-p_4)(1-p_5)p_7(1-p_8).$$

Let $Pr[s \mid F]$ denote the conditional probability of system state s given that s is drawn from the set of states F.

Then

$$Pr[s \mid F] = \begin{cases} \dfrac{Pr[s]}{Pr[F]} & s \in F \\ 0 & s \notin F \end{cases} \tag{1.2}$$

For example, $Pr[(0, 0, 1, 1, 1, 1, 0, 1) \mid (0, *, 1, 1, 1, *, 0, 1)]$ is

$$\frac{p_1p_2(1-p_3)(1-p_4)(1-p_5)(1-p_6)p_7(1-p_8)}{p_1(1-p_3)(1-p_4)(1-p_5)p_7(1-p_8)} = p_2(1-p_6)$$

In network reliability and many other types of problems the n-component system has a monotonic property. Define a partial order $\subseteq$ on system states as follows:

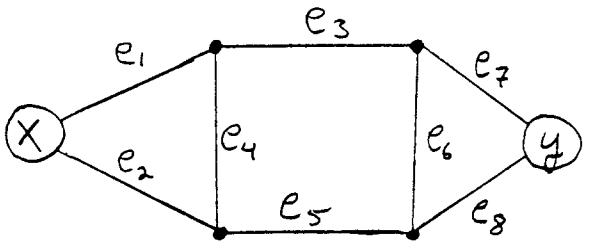
$s \subseteq t$ *iff* the set of components failing in system state s is a subset of the set of components failing in state t.

An n-component system is monotone if for all failure states s and system states t, $s \subseteq t$ implies t is also a failure state. In any monotonic n-component system the set of failure states can be represented by failure sets which correspond naturally to minimal failure states (minimal with respect to the partial order $\subseteq$). If s is a minimal failure state, then the corresponding failure set F is the set of all states t such that $s \subseteq t$. Thus, F can be written as $(c_1, \ldots, c_n)$

where $c_i = 0$ if component i is failing in state s, and $c_i = *$ if component i is working in state s.

2. A Network Example

Figure 1 shows an undirected network with eight edges and two designated vertices, x and y. Each edge e_i is failing with probability p_i and working with probability $1 - p_i$. The network is said to fail if there is no path of working edges between x and y.



$$(p_1, p_2, \ldots, p_8) = (.1, .5, .4, .3, .2, .4, .1, .2)$$

Figure 1 - *Two - Terminal Reliability Problem*

An $x - y$ cut set is a minimal set of edges whose deletion leaves no path between x and y. Then the network fails if an only if all the edges in some $x - y$ cut set fail, and thus the set of failure states of the system can be expressed as the union of failure sets which correspond to the $x - y$ cuts. These failure sets are listed in Table 2, where the probability of each failure set is also given.

k	8-tuple representing failure set F_k								$Pr(F_k)$
	1	2	3	4	5	6	7	8	
1	*	*	*	*	*	*	0	0	.02
2	*	*	*	*	0	0	0	*	.008
3	0	*	*	0	*	0	*	0	.0024
4	*	*	0	*	*	0	*	0	.032
5	*	*	0	*	0	*	*	*	.08
6	0	*	*	0	0	*	*	*	.006
7	*	0	*	0	*	0	0	*	.006
8	*	0	0	0	*	*	*	*	.06
9	0	0	*	*	*	*	*	*	.05

Table 2 - *List of Failure Sets for Network of Figure 1*

3. Monte-Carlo Area Estimation

In this section we present a Monte-Carlo technique to estimate the area of a region in the Euclidean plane. Most of the Monte-Carlo algorithms presented in this paper for the estimation of the failure probability of a system are analogous to this area estimation technique, and the explanation of the algorithms will rely heavily upon this analogy.

Suppose a region E of known area $A(E)$ encloses the region U of unknown area in the plane. Furthermore, suppose region E is subdivided into b blocks such that the area of each block i is known to be a_i, thus $A(E) = \sum_{i=1}^{b} a_i$. Suppose the region U consists of some subset of these blocks. Let α_i indicate whether or not block i is in region U, i.e.,

$$\alpha_i = \begin{cases} 1 & \text{if block } i \in U \\ 0 & \text{if block } i \notin U \end{cases}$$

Then the area of region U, $A(U)$, can be written as $\sum_{i=1}^{b} a_i \cdot \alpha_i$.

A straightforward method for determining $A(U)$ is to compute the above sum, but if b is large, this is a costly calculation. In the applications we consider b is very large.

Suppose instead we have a method to randomly select block i out of the set of all blocks with probability $\frac{\alpha_i}{A(E)}$. An unbiased estimator, Y, of the quantity $A(U)$ can be generated by randomly selecting block i with probability $\frac{\alpha_i}{A(E)}$ and letting $Y = \alpha_i \cdot A(E)$. The expected value of Y, $E[Y]$, is $\sum_{i=1}^{b} \frac{\alpha_i}{A(E)} \alpha_i \cdot A(E) = A(U)$.

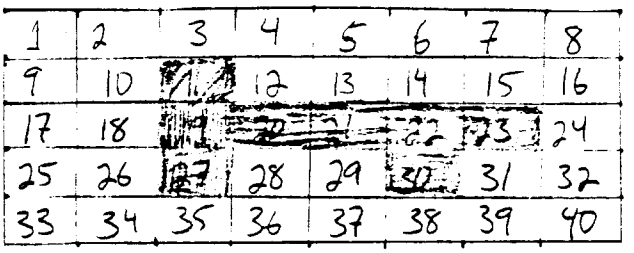


Figure 3 - *E is entire region*
U is the shaded region, E is subdivided into 40 blocks

$\alpha_5 = 0$, $\alpha_{11} = 1$.

The algorithm can be repeated many times yielding estimator Y_j in the j^{th} trial. $\bar{Y} = \frac{(Y_1 + \ldots + Y_N)}{N}$ is an unbiased estimator of $A(U)$.

We will show in the next section that the number of trials necessary to guarantee a specified degree of accuracy and confidence in the estimator is linearly proportional to $\frac{A(E)}{A(U)}$. It is therefore desirable that this ratio be small.

4. Convergence of Monte-Carlo Algorithms

Suppose the Monte-Carlo algorithm is repeated N times. Let Y_i be the value of the estimator obtained from the t^{th} trial. Let $\bar{Y} = \frac{(Y_1 + Y_2 + \ldots + Y_N)}{N}$. A meaningful measure of the quality of the estimator $\bar{Y}$ is its relative error, given by

$$\left| \frac{\bar{Y} - u}{u} \right|$$

where u is the expected value of the estimator produced by the algorithm (u is the quantity we are trying to estimate). We next derive an upper bound on the number of trials N required to guarantee that the relative error will exceed a specified value ε with probability less than or equal to a specified value δ. For example, if we specify $\varepsilon = .05$ and $\delta = .1$, we are requiring N to be large enough that the relative error will be greater than 5% no more than 10% of the time. For the sake of brevity, a Monte-Carlo algorithm will be called an (ε, δ) algorithm if the algorithm achieves these guarantees.

Let σ^2 be the variance of Y, where Y is the value obtained in a single Monte-Carlo trial. Then the variance of $\tilde{Y}$ is $\frac{\sigma^2}{N}$. By Chebyshev's inequality

$$Pr\left[\left|\frac{\tilde{Y}-u}{u}\right| > \varepsilon\right] = Pr\left[\,|\,\tilde{Y}-u\,| > \varepsilon u\,\right] \leq \frac{\sigma^2}{N\varepsilon^2 u^2}.$$

Thus, in order that $Pr\left[\left|\frac{\tilde{Y}-u}{u}\right| > \varepsilon\right]$ be less than or equal to δ, it suffices that

$$N \geq \frac{\sigma^2}{u^2} \cdot \frac{1}{\delta\varepsilon^2}.$$

Notice that there are two factors in the right-hand side of the inequality: $\frac{\sigma^2}{u^2}$ which depends on the Monte-Carlo algorithm and problem instance; and $\frac{1}{\delta\cdot\varepsilon^2}$ which depends on the desired relative accuracy of $\tilde{Y}$ and the desired confidence level of obtaining this accuracy. The rest of this analysis will only be concerned with $\frac{\sigma^2}{u^2}$.

For the area estimation algorithm presented in the previous section the random variable Y, which is the estimator of the area of region U, $A(U)$, is a Bernoulli random variable multiplied by the area of region E, $A(E)$. Thus

$\sigma^2 = A(E) \cdot A(U) - A(U)^2$ and thus

$$\frac{\sigma^2}{u^2} = \frac{A(E)}{A(U)} - 1.$$

The number of trials, N, necessary to achieve an (ε, δ) algorithm is

$$\left[\frac{A(E)}{A(U)} - 1\right] \cdot \frac{1}{\delta\varepsilon^2}. \tag{4.1}$$

We note that since Chebychev's inequality is true for any probability distribution this may be a very conservative upper bound on the number of trials.

If the area of region U is not much smaller than the area of region A, then the number of trials necessary is small. Our goal is to design an algorithm such that $\frac{A(E)}{A(U)}$ is small. We first present a standard algorithm to estimate the failure probability of an n-component system which can be viewed as an area estimation algorithm where for n-component systems typically encountered in practice, the ratio $\frac{A(E)}{A(U)}$ is very large.

5. Straight Simulation Monte-Carlo Method

A simple Monte-Carlo algorithm to estimate $Pr[S]$, the probability that the n-component system is in a failure state, follows.

Step 1 randomly select system state s with probability $Pr[s]$

Step 2 the estimator Y of $Pr[S]$ is $\begin{cases} 1 \text{ if } s \in S \\ 0 \text{ otherwise} \end{cases}$

The analogy to the area estimation technique goes as follows. The set of all system states corresponds to the enclosing region E. The system states correspond to the blocks into which the enclosing region is subdivided, where the area of each system state s is $Pr[s]$ and hence $A(E) = 1$. Region U comprises the set of all failure states S and hence the area of region U is

$Pr[S]$. The expected value of Y is equal to $Pr[S]$, however if $Pr[S]$ is small compared to one (which is typical of n-component systems) the number of trials must be very large to estimate $Pr[S]$ accurately.

The motivation for this work is to design a Monte-Carlo algorithm which estimates $Pr[S]$ accurately with a small number of trials even when $Pr[S]$ is very small.

6. A Description of the Coverage Algorithm

Assume that S, the set of failure states of an n-component system, is specified as the union of failure sets $F_1, F_2, \ldots, F_m$. Then the failure probability of the system is given by

$$Pr[S] = Pr[\bigcup_{k=1}^{m} F_k].$$

We noted in Section 2 that it is easy to compute $Pr[F_k]$ where F_k is any one of the m failure sets. If the m failure sets were disjoint, then calculating $Pr[S]$ would be simply a matter of computing $\sum_{i=1}^{m} Pr[F_k]$. Unfortunately, the failure sets are not disjoint in general. Furthermore, the classical formulas for the probability of a union of sets do not lead to efficient algorithms for evaluating $Pr[S]$. For example, the inclusion-exclusion formula

$$Pr[S] = Pr[\bigcup_{k=1}^{m} F_k] = \sum_{k=1}^{m} Pr[F_k] - \sum_{k_1=1}^{m} \sum_{k_2=1}^{k_1-1} Pr[F_{k_1} \cap F_{k_2}]$$

$$+ \sum_{k_1=1}^{m} \sum_{k_2=1}^{k_1-1} \sum_{k_3=1}^{k_2-1} Pr[F_{k_1} \cap F_{k_2} \cap F_{k_3}] + \ldots + (-1)^{m+1} Pr[F_1 \cap F_2 \cap \ldots \cap F_m]$$

entails $2^m - 1$ terms. The terms can fluctuate wildly in value, making it impossible in general to obtain a good approximation by truncating the expansion after the first few terms.

Another well-known formula is

$$Pr[S] = Pr[\bigcup_{k=1}^{m} F_k] = Pr[F_1]$$

$$+ Pr[F_2 \cap \overline{F_1}] + Pr[F_3 \cap (\overline{F_1 \cup F_2})] + \ldots + Pr[F_m \cap (\overline{F_1 \cup \ldots \cup F_{m-1}})] .$$

This expansion has only m terms, but the individual terms seem hard to compute, and the most obvious algorithms based on this formula require a number of steps exponential in m. In fact, to compute $Pr[S]$ exactly is NP-hard [1].

We will not attempt to compute $Pr[S]$ exactly. Instead, the ability to easily calculate $\sum_{i=1}^{k} Pr[F_k]$ will be used to estimate $Pr[\bigcup_{k=1}^{m} F_k]$. Let cov(s), the coverage of failure state s, be the number of failure sets containing s. For example, in the network reliability problem described in Figure 1 and Table 2, the coverage of failure state $s = (1, 0, 1, 0, 0, 0, 0, 0)$ is three because s is contained in the failure sets F_1, F_2, and F_7. As a second example, suppose the set of failure states is the union of three failure sets shown in Figure 4. Then

$$S = F_1 \cup F_2 \cup F_3, \; cov(s_1) = cov(s_2) = 1, \; cov(s_3) = 2, \text{ and } cov(s_4) = 3.$$

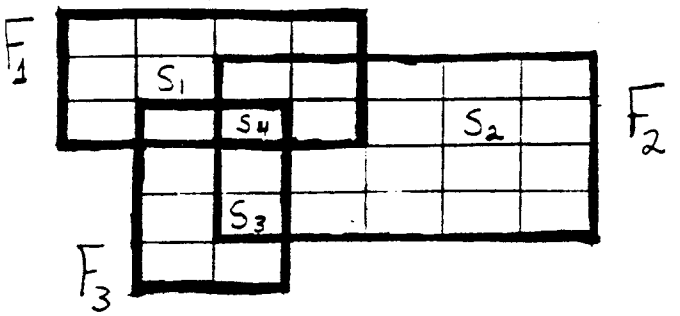


Figure 4 - *A representation of a set of failure states S as* $F_1 \cup F_2 \cup F_3$

Notice that each failure state s contributes a total of $cov(s) \cdot Pr[s]$ to $\sum_{k=1}^{m} Pr[F_k]$, because s contributes $Pr[s]$ to $Pr[F_k]$ for each F_k containing s. If instead we could arrange that each failure state s contributes a total of

$Pr[s]$,

then the total contribution of all failure states would be $Pr[S]$.

The new algorithm, called the coverage algorithm, is analogous to the area estimation algorithm. The area of the enclosing region E is $\sum_{k=1}^{m} Pr[F_k]$. The blocks that comprise region E are all ordered pairs (s, k), where s is a failure state contained in failure set F_k. Thus, failure state s will appear as the first component in exactly $cov(s)$ blocks. We let the area of each block (s, k), denoted $a(s, k)$, be equal to $Pr[s]$. Thus, the total area of all blocks in which s is the first component is $cov(s) \cdot Pr[s]$, and the total area of all blocks is indeed $\sum_{k=1}^{m} Pr[F_k]$.

Now we define region U, whose total area will be $Pr[S]$. For each failure state s we let exactly one the $cov(s)$ blocks in which s is the first component be in the region U. Thus, $\alpha(s, k) = 1$ for exactly one of the $cov(s)$ blocks in which s is the first component and $\alpha(s, k) = 0$ for the other $cov(s) - 1$ such blocks. Notice that it does not matter which of the $cov(s)$ blocks is in the region U. This makes it possible to select any one of the $cov(s)$ blocks in which s is the first component to be in region U.

Figure 5 illustrates the sample space for the set S of failure states shown in Figure 4. The region U is shaded in this figure. In this example block (s, k) is in region U if F_k is the smallest indexed failure set containing s.

How do we randomly select block (s, k) with probability $\frac{Pr[s]}{\sum_{k=1}^{m} Pr[F_k]}$?

This is a two-step process. First, we randomly select failure set F_k with probability $\frac{Pr[F_k]}{\sum_{k=1}^{m} Pr[F_k]}$. This is easy to do once the probability of each failure set has

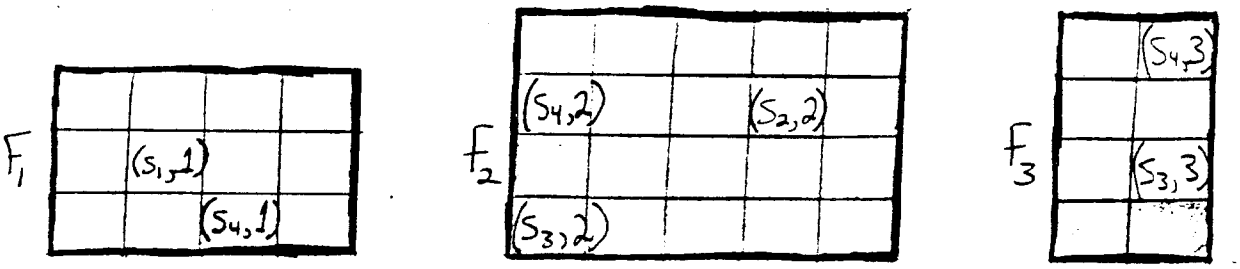


Figure 5 - *Sample space for S shown in Figure 4.*

been computed. This selects the second component k of the block. Then we randomly select a failure state s from failure set F_k with probability $\frac{Pr[s]}{Pr[F_k]}$. This is also easy to do, as we discuss in the next section. This selection picks the first component s of the block. Notice that this two-step process picks block (s, k) with probability $\frac{Pr[s]}{\sum_{k=1}^{m} Pr[F_k]}$.

The computation of $\alpha(s, k)$ is discussed in detail in the following sections. One trial of the coverage algorithm randomly selects block (s, k) by this two-step process, and returns $\alpha(s, k) \cdot \sum_{k=1}^{m} Pr[F_k]$ as the estimator of $Pr[S]$.

The advantage of this algorithm over the straight simulation algorithm is the number of trials, N, necessary to achieve an (ε, δ) algorithm. Recall from Formula (4.1) that for the area estimation algorithm, if N is greater than or equal to

$$\frac{\left\lceil \frac{A(E)}{A(U)} - 1 \right\rceil}{\delta \varepsilon^2} \tag{6.1}$$

then the algorithm is an (ε, δ) algorithm. For the straight simulation algorithm this value of N is

$$\frac{\left\lceil \frac{1}{Pr[S]} - 1 \right\rceil}{\delta \varepsilon^2} \tag{6.2}$$

But Formula (6.2) involves $Pr[S]$, the quantity the algorithm is attempting to estimate. There can be no upper bound on N derived from the formula since $Pr[S]$ can be arbitrarily small. Thus, without any prior information about $Pr[S]$ it is impossible, using only this formula, to put an upper bound on N a priori which will guarantee an (ε, δ) algorithm.

Let r_i be the probability that a state s selected from among all 2^n states with probability $Pr[S]$ has $cov(s)$ equal to i, i.e.

$$r_i = \sum_{\substack{s \text{ a state s.t.} \\ cov(s)=i}} Pr[s] \tag{6.3}$$

Thus,

$$A(U) = Pr[S] = \sum_{i=1}^{m} r_i \tag{6.4}$$

and

$$A(E) = \sum_{i=1}^{m} Pr[F_i] = \sum_{i=1}^{m} i \cdot r_i \tag{6.5}$$

The new algorithm is an (ε, δ) algorithm if the number of trials is

$$\frac{\left\lceil \frac{A(E)}{A(U)} - 1 \right\rceil}{\delta \cdot \varepsilon^2} = \frac{\left\lceil \frac{\sum_{i=1}^{m} i \cdot r_i}{\sum_{i=1}^{m} r_i} - 1 \right\rceil}{\delta \cdot \varepsilon^2} \tag{6.6}$$

But,

$$\frac{\sum_{i=1}^{m} i \cdot r_i}{\sum_{i=1}^{m} r_i} \leq m \tag{6.7}$$

Thus, if we let $N = \frac{m}{\delta\, \varepsilon^2}$ for the new algorithm we will have an (ε, δ) algorithm. Since m is known before we run the algorithm it is possible to put an a priori

upper bound on the number of trials necessary to guarantee an (ε, δ) algorithm. An even tighter upper bound is derived in Section 13.

Through the insight gained by this new algorithm, we are able to derive an easily computable upper bound on the number of trials to perform to guarantee on (ε, δ) algorithm for the straight simulation algorithm. The reasoning goes as follows: Equations (6.7), (6.5) and (6.4) imply that Formula (6.2) is less than or equal to

$$\frac{m}{\sum_{k=1}^{m} Pr[F_k]} \cdot \frac{1}{\delta\, \varepsilon^2} \tag{6.8}$$

Thus, the straight simulation algorithm is an (ε, δ) algorithm if the number of trials is given by to Formula (6.8).

We now compare the upper bounds on the number of trials derived for the straight simulation algorithm and the coverage algorithm. Let N be the upper bound on the number of trials for the straight simulation algorithm and $\tilde{N}$ be the upper bound on the number of trials for the coverage algorithm. We see that

$$\tilde{N} = N \cdot \sum_{k=1}^{m} Pr[F_k] \tag{6.9}$$

If $\sum_{k=1}^{m} Pr[F_k]$ is less than one, the upper bound of the number of trials to achieve an (ε, δ) algorithm for the new algorithm is less than the upper bound on the number of trials to achieve an (ε, δ) algorithm for the straight simulation algorithm. If $\sum_{k=1}^{m} Pr[F_k]$ is greater than one, the reverse inequality holds. Thus, a decision about which algorithm to use can be made based on the value of $\sum_{k=1}^{m} Pr[F_k]$. We expect $\sum_{k=1}^{m} Pr[F_k] \ll 1$ when the component failure probabilities are small, and therefore the upper bound on the number of trials for the

new algorithm will be substantially less than the number of trials for the straight simulation algorithm.

7. An Implementation of the Coverage Algorithm

We next present an implementation of the new Monte-Carlo algorithm. The input to the algorithm is an n-component reliability problem in the format described previously. The output from one trial of the algorithm is a number which is an unbiased estimator of the system failure probability.

Preprocessing

For $k = 1, 2, \ldots, m$ compute $Pr[F_k]$.

If F_k is represented by the array $(c_1, \ldots, c_n)$ then, as described previously, $Pr[F_k] = \prod_{i=1}^{n} p_i^{c'_i}(1-p_i)^{c''_i}$

Allocate an array FS of size m.

For $k = 1, 2, \ldots, m$, $FS[k] \leftarrow \dfrac{\sum_{j=1}^{k} Pr[F_j]}{\sum_{j=1}^{m} Pr[F_j]}$

Array FS will be used to randomly select failure set F_k with probability $\dfrac{Pr[F_k]}{\sum_{j=1}^{m} Pr[F_j]}$.

Monte-Carlo Trial

Step 1 Randomly select a failure set F_k with probability $\dfrac{Pr[F_k]}{\sum_{j=1}^{m} Pr[F_j]}$. This can be done by picking a random number r from the uniform distribution over [0, 1] and determining

$k = \min \{ j \mid FS[j] \geq r \}$ using binary search.

Step 2 Randomly select $s \in F_k$ with probability $Pr[s \mid F_k] = \frac{Pr[s]}{Pr[F_k]}$.

If F_k is specified by the array $(c_1, c_2, \ldots, c_n)$ then $s = (b_1, b_2, \ldots, b_n)$ is chosen as follows:

if $c_i = 0$, then $b_i = 0$
if $c_i = 1$, then $b_i = 1$
if $c_i = *$, $\begin{cases} \text{then choose } b_i = 0 \text{ with probability } p_i \\ \quad \text{and } b_i = 1 \text{ with probability } 1 - p_i \end{cases}$

At this point block (s, k) has been selected.

Step 3 Compute $\alpha(s, k)$

$$\text{Define } \alpha(s, k) = \begin{cases} 1 \text{ if } F_k \text{ is the smallest indexed} \\ \qquad \text{failure set such that } s \in F_k \\ 0 \text{ otherwise} \end{cases}$$

Then $\alpha(s, k)$ can be computed by finding the smallest index i such that $s \in F_i$. If $k = i$ then $\alpha(s, k) = 1$, otherwise $\alpha(s, k) = 0$.

Step 4 The estimator, Y, of $Pr[S]$ is $\alpha(s, k) \cdot \sum_{k=1}^{m} Pr[F_k]$.

The time to perform the preprocessing step given the list of failure sets $F_1, \ldots, F_m$ is $O(m \cdot n)$. For the two-terminal problem the failure sets could be given implicitly by a data structure representing the graph. The preprocessing step consists of listing all the $x - y$ cuts in the graph and then proceeding as before. All of the cuts in the graph can be listed in time $O(m \cdot n)$ [2] (in this case m = # cuts in the graph, n = # edges in the graph), so the total preprocessing time is $O(m \cdot n)$ in this case also.

Step 1 of the trial takes time $O(\log m)$ using binary search to find k. Since there are 2^n system states, there can be at most 2^n failure sets, hence the time for Step 1 is $O(n)$. Step 2 also takes time $O(n)$, and Step 4 can be performed in constant time.

In this implementation the computation of $\alpha(s, k)$ in Step 3 takes time at most $O(m \cdot n)$. This can be seen as follows: $\alpha(s, k)$ is computed by sequentially searching through the failure sets until we find a failure set F_i such that $s \in F_i$, and then $\alpha(s, k) = 1$ if $i = k$, otherwise $\alpha(s, k) = 0$. Each test for membership of s in a failure set takes $O(n)$ time. In the worst case all m failure sets will be examined, thus the total running time is $O(m \cdot n)$.

In Section 6 we found that $\frac{m}{\delta \varepsilon^2}$ trials are sufficient to achieve an (ε, δ) algorithm. Thus, the running time for all the trials is

$$O\left[\frac{m^2 n}{\delta \varepsilon^2}\right]. \tag{7.1}$$

The running time per trial is dominated by the time to compute $\alpha(s, k)$ in Step 3 of the algorithm. In the following sections we will discuss methods to substantially reduce the running time of the algorithm based on alternative ways to compute α. First, we will generalize the definition of α in a way that helps us compute α quickly.

8. A Generalization of the Coverage Algorithm

The requirement that $\alpha(s, k) = 1$ for exactly one of the $cov(s)$ blocks in which s is the first component and $\alpha(s, k) = 0$ for the other $cov(s) - 1$ such blocks can be relaxed. A more general scheme is to allow $\alpha(s, k)$ to be a random variable such that

$$\sum_{\{k \mid s \in F_k\}} E[\alpha(s, k)] = 1. \tag{8.1}$$

Any such scheme can be viewed as a probabilistic allocation of the probability of system state s to the set of blocks in which s is the first component. Any allocation scheme fulfilling these more general requirements will produce an unbiased estimator of $Pr[S]$. As an example, letting $\alpha(s, k) = \frac{1}{cov(s)}$ for all blocks in

which s is the first component, fulfills these requirements. This particular allocation scheme has the smallest variance among all allocation schemes, which can be seen as follows. The variance σ^2 is equal to $E[Y^2] - E[Y]^2$, but, since $E[Y]^2 = Pr[S]^2$ for any choice of α, the allocation which minimizes $E[Y^2]$ will have the smallest variance. Now,

$$E[Y^2] = \left[\sum_{k=1}^{m} Pr[F_k]\right]\left[\sum_{s \in F_k} Pr[s] \cdot \left[\sum_{\{k \mid s \in F_k\}} E[\alpha^2(s, k)]\right]\right]. \qquad (8.2)$$

The choice which minimizes $\sum_{\{k \mid s \in F_k\}} E[\alpha^2(s, k)]$ subject to $\sum_{\{k \mid s \in F_k\}} E[\alpha(s, k)] = 1$ will minimize the variance. A little algebraic manipulation shows that this is minimized when $\alpha(s, k) = \frac{1}{cov(s)}$ for all blocks (s, k) in which s is the first component.

9. A Hybrid Allocation Scheme - The Cutoff Method

Let c, the cutoff, be a positive integer. We will allocate the probability of failure state s among the blocks in which s is the first component as follows:

1.) If $cov(s) \leq c$ then allocate $Pr[s]$ equally among all $cov(s)$ blocks, i.e., $\alpha(s, k) = \frac{1}{cov(s)}$ for all $cov(s)$ such blocks.

2.) If $cov(s) > c$ then allocate $Pr[s]$ equally among c of the blocks, i.e., $\alpha(s, k) = \frac{1}{c}$ for c of the blocks and $\alpha(s, k) = 0$ for the other $cov(s) - c$ such blocks.

The reason that c is called the cutoff is because in the implementation of the hybrid allocation scheme the algorithm finds the first $\min\{c, cov(s)\}$ failure sets that contain s. The probability of state s is allocated equally among the blocks in which the second component is the index of one of these $\min\{c, cov(s)\}$ failure sets. Thus, $\alpha(s, k) = \frac{1}{\min\{c, cov(s)\}}$ if k is the index

of one of these failure sets, otherwise $\alpha(s, k) = 0$. The value c is an upper bound, or cutoff, on the number of failure sets containing s that the algorithm must find in order to compute $\alpha(s, k)$.

When c is infinite then $\alpha(s, k) = \frac{1}{cov(s)}$ for all blocks (s, k); this is the minimum-variance case. When c is one then $\alpha(s, k) = 1$ for exactly one of the $cov(s)$ blocks in which s is the first component and $\alpha(s, k) = 0$ for the other $cov(s) - 1$ such blocks; this is the maximum-variance case for the hybrid method. Recall from formulas (6.3), (6.4) and (6.5) the definition of r_i. Then $\frac{i r_i}{\sum_{i=1}^{m} i r_i}$ is the probability a system state with coverage i is randomly selected in one trial of the algorithm. Thus, for the cutoff method $\frac{\sigma^2}{u^2}$ can be expressed as

$$\left(\frac{\sum_{1 \le i < c} \frac{r_i}{i} + \frac{1}{c} \sum_{i \ge c} r_i}{\sum_{i \ge 1} r_i} \right) \left(\frac{\sum_{i \ge 1} i r_i}{\sum_{i \ge 1} r_i} \right) - 1 \qquad (9.1)$$

It is trivial to modify the coverage algorithm presented in Section 7 to incorporate the hybrid allocation scheme. The only change is in the computation of $\alpha(s, k)$, which can be described as follows:

Step 3 Sequentially search through the failure sets until either c failure sets F_i are found such that $s \in F_i$ or all the failure sets are searched. Let l be the number of failure sets found such that $s \in F_i$, then $l = \min(c, cov(s))$. If k is the index of one of the l failure sets found, then $\alpha(s, k) = \frac{1}{l}$, otherwise $\alpha(s, k) = 0$.

If we let $c = \infty$ (in which case $\alpha(s, k) = \frac{1}{cov(s)}$), then the time per trial is still $O(m \cdot n)$. The best upper bound we can prove on the number of trials necessary to achieve an (ε, δ) algorithm is still $\frac{m}{\delta \varepsilon^2}$, so the total running time

is still $O\left\{\frac{m^2 n}{\delta \varepsilon^2}\right\}$. However, using the algorithm with $c = \infty$ will result in a stochastically better estimate of $Pr[S]$ then using the algorithm with a smaller value of c for the same number of trials.

10. A Substantially Faster Variation of the Coverage Algorithm

In this section we present an alternative implementation of the coverage algorithm presented in Section 7. We will prove that an upper bound on the running time of this new algorithm to guarantee an (ε,δ) algorithm is $O\left\{\frac{m \cdot n}{\delta \varepsilon^2}\right\}$.

Recall that for the previous implementation of the coverage algorithm we were able to prove an upper bound on the running time of $O\left\{\frac{m^2 n}{\delta \varepsilon^2}\right\}$. Since m is typically very large in comparison to n, this improvement in the running time is substantial. We call the new algorithm the linear time coverage algorithm to emphasis the fact that the running time is linear in the input size (which is $m \cdot n$) divided by $\delta \cdot \varepsilon^2$.

We assume the most general input format. The input consists of the failure probabilities of the n components and a list of the m failure sets in the format described in the first section of this paper.

We first present the algorithm. The preprocessing step is exactly the same as it is for the coverage algorithm. The first two steps, randomly selecting block (s, k) with probability $\frac{Pr[s]}{\sum_{k=1}^{m} Pr[F_k]}$, are also exactly the same as they are for the coverage algorithm. Once block (s, k) has been selected, the linear time coverage algorithm produces two unbiased estimators, $\alpha(s, k)$ and $\alpha'(s, k)$, of $\frac{1}{cov(s)}$ in Step 3 which are independent of one another. Note that both $\sum_{\{k \mid s \in F_k\}} E[\alpha(s, k)] = 1$ and $\sum_{\{k \mid s \in F_k\}} E[\alpha'(s, k)] = 1$. Thus, either

$Y = \alpha(s, k) \sum_{k=1}^{m} Pr[F_k]$ or $Y' = \alpha'(s, k) \sum_{k=1}^{m} Pr[F_k]$ will be an unbiased estimator of $Pr[S]$.

A Description of the Linear Time Coverage Algorithm

$\tilde{Y} \leftarrow 0$, $\tilde{Y}' \leftarrow 0$

numtrials $\leftarrow 0$

time $\leftarrow 0$

Repeat steps 1-5 until *time* $= \dfrac{c \cdot m}{\delta \cdot \varepsilon^2}$

Step 1 randomly select a failure set F_k with probability $\dfrac{Pr[F_k]}{\sum_{k=1}^{m} Pr[F_k]}$ as before

Step 2 randomly select $s \in F_k$ with probability $\dfrac{Pr[s]}{Pr[F_k]}$ as before

Step 3 $l \leftarrow 0$

Do until a failure set F_i is selected such that $s \in F_i$

randomly select failure set F_i with probability $\dfrac{1}{m}$
$l \leftarrow l + 1$
Check to see if $s \in F_i$ (*)
time $\leftarrow$ *time* + 1

$$\alpha(s, k) = \frac{l}{m}$$

$$\alpha'(s, k) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise} \end{cases}$$

Step 4 The estimators, Y & Y', of $Pr[S]$ are

$$Y = \alpha(s, k) \cdot \sum_{k=1}^{m} Pr[F_k]$$

$$Y' = \alpha'(s, k) \cdot \sum_{k=1}^{m} Pr[F_k]$$

Step 5 $numtrials \leftarrow numtrials + 1$, $\tilde{Y} \leftarrow \tilde{Y} + Y$, $\tilde{Y}' \leftarrow \tilde{Y}' + Y'$

Go to step 1

Step 6 Estimator 1 = $\dfrac{\tilde{Y}}{numtrials}$

Estimator 2 = $\dfrac{\tilde{Y}'}{numtrials}$

One trial of the algorithm is the execution of steps 1-5. The value of the constant c is discussed in the following theorem, which establishes that the choice $c = 16$ gives an (ε, δ) algorithm. The running time for each trial is dominated by the time to perform the test marked with a (*) in Step 3. This test takes $O(n)$ time to perform. The total running time is therefore $\frac{c \cdot m \cdot n}{\delta \cdot \varepsilon^2}$. To simplify notation, we call the number of times (*) is performed per trial the length of the trial.

Now we will show that $E[\alpha(s, k)] = \frac{1}{cov(s)}$. Suppose (s, k) is picked in Steps 1 and 2 of one trial. We are interested in computing

$$E[\alpha(s, k)] = \frac{E[\text{length of trial}]}{m}.$$

Each time line (*) is executed there is a chance of $\frac{cov(s)}{m}$ that $s \in F_i$ since each failure set is picked with probability $\frac{1}{m}$ and s is an element of $cov(s)$ failure sets. Let $X(s, k)$ be the length of the trial given that (s, k) is picked in Steps 1 and 2 of the trial. Then $X(s, k)$ is a random variable geometrically distributed with rate $\frac{cov(s)}{m}$, and

$$E[\alpha(s, k)] = \frac{E[X(s, k)]}{m} = \frac{m}{cov(s)} \cdot \frac{1}{m} = \frac{1}{cov(s)}. \qquad (10.3)$$

Now we will show that $E[\alpha'(s, k)] = \frac{1}{cov(s)}$. Suppose (s, k) is picked in

Steps 1 and 2 of one trial. Once the algorithm finds a failure set F_i such that $s \in F_i$, the probability that $i = k$ is exactly $\frac{1}{cov(s)}$ independently of the length of the trial. Thus,

$$E[\alpha'(s, k)] = \frac{1}{cov(s)} \tag{10.4}$$

and $\alpha(s, k)$ and $\alpha'(s, k)$ are independent estimators of $\frac{1}{cov(s)}$.

We now compute $\frac{E[Y^2]}{E[Y]^2}$ and $\frac{E[Y'^2]}{E[Y']^2}$, which we call $\underline{\vartheta}$ and $\underline{\vartheta'}$ in the following discussion to simplify notation. These formulas are needed to analyze the running time of the algorithm. Using the notation introduced in formulas (6.3),(6.4) and (6.5) we see that

$$E[Y^2] = \left[\sum_{i=1}^{m} i\,r_i\right]\left[\sum_{i=1}^{m} \frac{r_i}{i}\left(2 - \frac{i}{m}\right)\right] \leq 2\left[\sum_{i=1}^{m} i\,r_i\right]\left[\sum_{i=1}^{m} r_i\right] \tag{10.5}$$

and

$$E[Y'^2] = \left[\sum_{i=1}^{m} i\,r_i\right]\left[\sum_{i=1}^{m} r_i\right]. \tag{10.6}$$

Thus, if we let

$$\mu = \frac{Pr[S]}{\sum_{i=1}^{m} Pr[F_k]} = \frac{\sum_{i=1}^{m} r_i}{\sum_{i=1}^{m} i \cdot r_i} \tag{10.7}$$

we see that

$$\vartheta \leq \frac{2}{\mu} \tag{10.8}$$

and

$$\vartheta' \leq \frac{1}{\mu} \tag{10.9}$$

The standard technique to guarantee an (ε, δ) algorithm is to compute an a priori upper bound on the number of trials sufficient for an (ε, δ) algorithm. Typically Chebyshev's inequality is used to compute an upper bound on the number of trials sufficient to guarantee an (ε, δ) algorithm. Instead, we put an upper bound on the number of times (*) in Step 3 must be executed to guarantee an (ε, δ) algorithm. We will prove that if (*) is executed $\frac{c \cdot m}{\delta \cdot \varepsilon^2}$ times during the course of the algorithm, we have an (ε, δ) algorithm (where c is a suitably chosen constant ≥ 1). The intuitive reason why this type of time bound will guarantee an (ε, δ) algorithm follows:

$$E[\text{length of a trial}] = m\,\mu\,.$$

If (*) is executed $t = \frac{c\,m}{\delta \cdot \varepsilon^2}$ times, then the expected number of trials completed by time t is approximately $\frac{t}{m\,\mu} = \frac{c}{\mu\,\delta \cdot \varepsilon^2}$. The upper bound on both ϑ and ϑ' can be expressed as $\frac{\bar{c}}{\mu}$ (where $\bar{c} = 2$ and 1, respectively). Thus, if the estimator Y (or Y') for each trial were independent of the length of the trial and if c were chosen to be suitably larger than $\bar{c}$, then an application of Chebyshev's inequality would give us the desired result. Since the estimator Y (or Y') from each trial depends on the length of the trial, we will use Kolmogorov's inequality (which is a stronger version of Chebyshev's inequality) to prove the result.

We first introduce some notation which will simplify the proof that executing (*) in Step 3 $O\left(\frac{m}{\delta\varepsilon^2}\right)$ times will guarantee an (ε,δ) algorithm. Let X_j be a random variable denoting the length of the j^{th} trial. Thus, $\{X_j\}$ is a sequence of i.i.d. random variables where

$$E[X_j] = \sum_{i=1}^{m} \frac{i \cdot r_i}{\sum_{i=1}^{m} i \cdot r_i} \left(\frac{m}{i}\right) = m\,\frac{Pr[S]}{\sum_{i=1}^{m} Pr[F_k]} = m\,\mu\,. \qquad (10.10)$$

The estimator Y_j (or Y_j') of $Pr[S]$ generated at the end of the j^{th} trial is also a random variable such that $E[Y_j]$ (or $E[Y_j']$) = $Pr[S]$. Y_j is not independent of X_j, but the sequence of ordered pairs $\{(X_j, Y_j)\}$ are independently and identically distributed (readers familiar with renewal-reward theory will recognize that this is a renewal-reward process).

We let $S_n = \sum_{i=1}^{n} X_i$ be the time at which the n^{th} trial is completed, $R_n = \sum_{i=1}^{n} Y_i$ ($R_n' = \sum_{i=1}^{n} Y_i'$) be the sum of the estimates from the first n trials and $N(t)$ be the number of trials completed by time t.

The running time of the algorithm will depend upon the upper bound on ϑ (ϑ') if the algorithm uses Y (Y') as the estimator of $Pr[S]$. If we let $\bar{c} = 2 \cdot (\bar{c} = 1)$, then $\vartheta \le \frac{\bar{c}}{\mu}$ $\left[\vartheta' \le \frac{\bar{c}}{\mu}\right]$. In the following discussion we express the running time in terms of $\bar{c}$ and use the variables Y and R in place of Y and R or Y' and R' to avoid proving two theorems depending upon whether Y or Y' is used as the estimator of $Pr[S]$.

Theorem: The Linear Time Coverage Algorithm is an (ε , δ) algorithm when the estimator used is Estimator 1 and c = 16, or when the estimator used is Estimator 2 and c = 8.

Proof : We will prove that

$$Pr\left[\left|\frac{\frac{R_{N(t)}}{N(t)} - E[Y]}{E[Y]}\right| \ge \varepsilon\right] \le \delta$$

when

$$t = \frac{4m\bar{c}}{\delta\,\varepsilon^2}\left(1 + \frac{\varepsilon^{\frac{2}{3}}}{2\bar{c}^{\frac{1}{3}}}\right)\left(1 + \frac{\varepsilon^{\frac{2}{3}}}{\bar{c}^{\frac{1}{3}}}\right)$$

$$E[X] = m\cdot\mu\,,\quad \frac{E[X^2]}{E[X]^2} \le \frac{2}{\mu},\quad \frac{E[Y^2]}{E[Y]^2} \le \frac{\bar{c}}{\mu}$$

Comment 1 : When $\bar{c} \ge 1$ and $\varepsilon \le .25$, $\left[1 + \frac{\varepsilon^{\frac{2}{3}}}{2\bar{c}^{\frac{1}{3}}}\right]\left[1 + \frac{\varepsilon^{\frac{2}{3}}}{\bar{c}^{\frac{1}{3}}}\right] \le 2$. Thus when Estimator 1 is used as the estimator ($\bar{c} = 2$), the algorithm is an (ε, δ) algorithm when $c = 16$, and when Estimator 2 is used as the estimator ($\bar{c} = 1$), the algorithm is an (ε, δ) algorithm when $c = 8$.

Comment 2 : $\frac{R_{N(t)}}{N(t)}$ is not an unbiased estimator of $Pr[S]$ because $N(t)$ is not a valid stopping time [3]. Nevertheless, the proof of the theorem is still valid. We could complete the trial in process at time t and use $\frac{R_{N(t)+1}}{N(t)+1}$ as the unbiased estimator of $Pr[S]$, but this would make the running time of the algorithm a random variable. We choose not to complete the trial in progress at time t and accept the small bias in the estimator. In the following discussion we use the term "stopping time" to mean the time the algorithm is stopped, we do not mean stopping time as it is defined technically in renewal theory.

Back to Proof : Fix $t' = \frac{d\,m}{\delta\,\varepsilon^2}$, $k = \frac{d}{\mu\,\delta\,\varepsilon^2}$ where d is a constant to be determined later. Let β be a constant (whose optimal value we will determine later) and let $t'' = t'(1 + \beta)$. First, we investigate what can be said using stopping time t''.

$$Pr\left[\left|\frac{\frac{R_{N(t'')}}{N(t'')} - E[Y]}{E[Y]}\right| \ge \varepsilon\right] =$$

$$Pr\left[\left|\frac{\frac{R_{N(t'')}}{N(t'')} - E[Y]}{E[Y]}\right| \geq \varepsilon \text{ and } N(t'') < k\right] + \tag{10.11}$$

$$Pr\left[\left|\frac{\frac{R_{N(t'')}}{N(t'')} - E[Y]}{E[Y]}\right| \geq \varepsilon \text{ and } N(t'') \geq k\right] \tag{10.12}$$

We will compute upper bounds on formula (10.11) and (10.12) separately.

Upper bound on formula (10.11)

$$Pr\left[\left|\frac{\frac{R_{N(t'')}}{N(t'')} - E[Y]}{E[Y]}\right| \geq \varepsilon \text{ and } N(t'') < k\right] \leq Pr[N(t'') < k] =$$

$$Pr[S_k > t''] = Pr\left[\frac{S_k}{k} > \frac{t'(1+\beta)}{k}\right] \leq$$

$$Pr\left[\left|\frac{\frac{S_k}{k} - \frac{t'}{k}}{\frac{t'}{k}}\right| > \beta\right] = Pr\left[\left|\frac{\frac{S_k}{k} - m\mu}{m\mu}\right| > \beta\right]$$

Let $\varepsilon' = \beta$, $\delta' = \frac{2\delta\varepsilon^2}{d\beta^2}$. Since $k = \frac{d}{\mu\delta\varepsilon^2} = \left(\frac{2}{\mu}\right)\frac{1}{\delta'\varepsilon'^2}$, we use Chebyshev's inequality to conclude that an upper bound on formula (10.11) is

$$\delta' = \frac{2\,\delta\,\varepsilon^2}{d\,\beta^2} \tag{10.13}$$

Upper bound on formula (10.12)

$$Pr\left[\left|\frac{\frac{R_{N(t'')}}{N(t'')} - E[Y]}{E[Y]}\right| \geq \varepsilon \text{ and } N(t'') \geq k\right] \leq$$

$$Pr\left[\left|\frac{\frac{R_r}{r} - E[Y]}{E[Y]}\right| \geq \varepsilon \text{ for some } r \geq k \text{ and } N(t'') \geq k\right] \leq$$

$$Pr\left[\left|\frac{\frac{R_r}{r} - E[Y]}{E[Y]}\right| \geq \varepsilon \text{ for some } r \geq k\right] \qquad (10.14)$$

We will use Kolmogorov's inequality to derive an upper bound on formula (10.14). We first state Kolmogorov's inequality [4] and then manipulate the inequality until it is in a form which is useful to derive an upper bound on formula (10.14).

Kolmogorov's Inequality

Let $Y_1, Y_2, \ldots, Y_n$ be independent random variables with the same distribution as Y such that $E[Y]$ and $\sigma^2[Y] = E[Y^2] - E[Y]^2$ are finite, and let $R_l = \sum_{i=1}^{l} Y_i$. For every $x > 0$,

$$Pr[\exists\, l \mid 1 \leq l \leq n \;\&\; |R_l - l \cdot E[Y]| > x\sqrt{n}\;\sigma[Y]] < \frac{1}{x^2} \qquad (10.15)$$

Substituting $\frac{\varepsilon n}{\sqrt{n}\;\sigma[Y]} E[Y]$ for x yields

$$Pr[\exists\, l \mid 1 \leq l \leq n \;\;\&\; |R_l - l \cdot E[Y]| > \varepsilon \cdot n \cdot E[Y]] < \frac{\sigma^2[Y]}{\varepsilon^2 \cdot n \cdot E[Y]^2}$$

$$\leq \frac{\bar{c}}{\mu \varepsilon^2 n} \quad \left[\text{Since } \frac{\sigma^2[Y]}{E[Y]^2} < \vartheta \text{ (or } \vartheta') \leq \frac{\bar{c}}{\mu}\right].$$

Once again, this can be rewritten as

$$Pr\left[\exists\, l \mid 1 \leq l \leq n \;\;\&\; \left|\frac{\frac{R_l}{l} - E[Y]}{E[Y]}\right| > \frac{\varepsilon \cdot n}{l}\right] < \frac{\bar{c}}{\mu \cdot \varepsilon^2 \cdot n}$$

Thus,

$$\frac{\bar{c}}{\mu \cdot \varepsilon^2 \cdot n \cdot 2^i} \geq Pr\left[\exists\, l \mid 1 \leq l < n \cdot 2^i \;\&\; \left|\frac{\frac{R_l}{l} - E[Y]}{E[Y]}\right| > \frac{\varepsilon \cdot n \cdot 2^i}{l}\right] \geq$$

$$Pr\left[\exists\, l \mid n \cdot 2^{i-1} \leq l < n \cdot 2^i \;\&\; \left|\frac{\frac{R_l}{l} - E[Y]}{E[Y]}\right| > \frac{\varepsilon \cdot n \cdot 2^i}{l}\right] \geq$$

$$Pr\left[\exists\, l \mid n\cdot 2^{i-1} \le l < n\cdot 2^{i} \;\&\; \left|\frac{\frac{R_l}{l} - E[Y]}{E[Y]}\right| > 2\varepsilon\right]$$

An upper bound on formula (10.14) can be derived as follows.

$$Pr\left[\left|\frac{\frac{R_r}{r} - E[Y]}{E[Y]}\right| \ge \varepsilon \text{ for some } r \ge k\right]$$

$$\le \sum_{i=1}^{\infty} Pr\left[\exists\, l \mid k\cdot 2^{i-1} \le l < k\cdot 2^{i} \;\&\; \left|\frac{\frac{R_l}{l} - E[Y]}{E[Y]}\right| \ge 2\cdot\left(\frac{\varepsilon}{2}\right)\right]$$

$$\le \frac{\bar{c}}{\mu\cdot\left(\frac{\varepsilon}{2}\right)^2\cdot k}\sum_{i=1}^{\infty}\frac{1}{2^i} = \frac{4\cdot\bar{c}}{\mu\cdot\varepsilon^2\cdot k} = \frac{4\cdot\bar{c}\cdot\delta}{d}$$

Thus, formulas (10.11) and (10.12) sum to less than

$$\frac{\delta}{d}\left(4\cdot\bar{c} + \frac{2\varepsilon^2}{\beta^2}\right) \tag{10.16}$$

Using stopping time $t'' = \frac{d\cdot m}{\delta\cdot\varepsilon^2}(1+\beta)$, we achieve an $\left(\varepsilon, \frac{\delta}{d}\left(4\cdot\bar{c} + \frac{2\varepsilon^2}{\beta^2}\right)\right)$ algorithm. If we substitute $\delta' = \delta\cdot\left(\frac{d}{4\cdot\bar{c} + 2\frac{\varepsilon^2}{\beta^2}}\right)$ for δ we achieve an (ε, δ) algorithm where now the stopping time is $\frac{d\cdot m}{\delta'\cdot\varepsilon^2}(1+\beta) = \frac{m}{\delta\cdot\varepsilon^2}\left(4\cdot\bar{c} + 2\cdot\frac{\varepsilon^2}{\beta^2}\right)(1+\beta)$.

The value of β which minimizes this stopping time when ε is small is $\beta = \sqrt[3]{\frac{\varepsilon^2}{\bar{c}}}$. Substituting this value for β yields

$$\frac{4m\,\bar{c}}{\delta\,\varepsilon^2}\left(1 + \frac{\varepsilon^{\frac{2}{3}}}{2\,\bar{c}^{\frac{1}{3}}}\right)\left(1 + \frac{\varepsilon^{\frac{2}{3}}}{c^{\frac{1}{3}}}\right).$$

This completes the proof.

The total running time of the linear time coverage algorithm is then $O(m \cdot n)$ for preprocessing plus $O\left[m \cdot \frac{n}{\delta \varepsilon^2}\right]$ time to execute (*) in Step 3 $O\left[\frac{m}{\delta \varepsilon^2}\right]$ times, which guarantees an (ε, δ) algorithm for the linear time coverage algorithm.

11. Two-Terminal Network Reliability Coverage Algorithm

In this section we show how the two-terminal network reliability problem can be attacked using a variant of the algorithm of Section 7. We give a fast way to compute α when the input to the algorithm is a list of the edge failure probabilities together with an adjacency list for the graph. The preprocessing step in this case consists of listing all the $x-y$ cuts in the graph [2].

The first two steps of one trial of this algorithm, picking block (s, k), are exactly the same as they are for all the previously described coverage algorithms. We will use the cutoff method described in Section 9 to compute $\alpha(s, k)$. Let c be the value of the cutoff. The adjacency list representation is used to list cut sets occurring among failing edges in state s. The algorithm lists cut sets occurring among failing edges in state s until either c cuts are found $(cov(s) \geq c)$ or until all $cov(s)$ cut sets occurring among failing edges in state s are found. $\alpha(s, k) = \frac{1}{\min\{c, cov(s)\}}$ if k is the index of one of the failure sets, otherwise $\alpha(s, k) = 0$.

The time for listing each cut set is $O(n)$ [2], thus the time to compute $\alpha(s, k)$ is

$$\min\{c, cov(s)\} \cdot n \,. \tag{11.1}$$

The average time to compute α for one trial of the algorithm is

$$\frac{\left[\sum_{1 \leq i < c} i^2 r_i + c \sum_{i \geq c} i\, r_i\right]}{\sum_{i \geq 1} i\, r_i} \cdot n \,. \tag{11.2}$$

When $c = 1$, the time per trial is $O(n)$. An upper bound on the number of trials sufficient to guarantee an (ε, δ) algorithm is

$$\left[\frac{\sum_{k=1}^{m} Pr[F_k]}{Pr[S]} - 1\right] \cdot \frac{1}{\delta \varepsilon^2} \leq \frac{m}{\delta \varepsilon^2}$$

as we saw in formula (6.1). Thus, an upper bound on the running time of $O\left[\frac{m \cdot n}{\delta \varepsilon^2}\right]$ is obtained if the number of trials is $\frac{m}{\delta \varepsilon^2}$ and $c = 1$.

In the next section we show that

$$\frac{\sum_{k=1}^{m} Pr[F_k]}{Pr[S]}$$

is less than or equal to $\prod_{i=1}^{n} (1 + p_i)$. Thus, if we execute

$$\frac{\left[\prod_{i=1}^{n} (1 + p_i) - 1\right]}{\delta \varepsilon^2}$$

trials, we have an (ε, δ) algorithm. If we let $c = 1$, the total running time is

$$O\left[m \cdot n + \frac{\left[\prod_{i=1}^{n} (1 + p_i) - 1\right] \cdot n}{\delta \varepsilon^2}\right].$$

12. An Upper Bound on the Number of Trails Necessary to Achieve an (ε, δ) Algorithm when the System is Monotonic

In this section we show for monotonic n-component systems

$$\frac{\sum_{k=1}^{m} Pr[F_k]}{Pr\left[\bigcup_{k=1}^{m} F_k\right]} \leq \prod_{i=1}^{n} (1 + p_i). \tag{12.1}$$

Thus, the number of trials sufficient to guarantee an (ε, δ) algorithm for

the coverage algorithms is $\dfrac{\left[\prod_{i=1}^{n}(1+p_i)-1\right]}{\delta\cdot\varepsilon^2}$ for monotonic n-component systems. Note that

$$\prod_{i=1}^{n}(1+p_i) \le e^{\sum_{i=1}^{n} p_i}.$$

Thus, as $\sum_{i=1}^{n} p_i$ goes to zero, the number of trials necessary also goes to zero. In marked contrast, since $Pr[S]$ goes to zero as $\sum_{i=1}^{n} p_i$ goes to zero, the number of trials necessary for the straight simulation method becomes unbounded as $\sum_{i=1}^{n} p_i$ goes to zero.

Note that $\prod_{i=1}^{n}(1+p_i)$ can be computed before any trials are performed. This calculation gives an a priori upper bound on the number of trials necessary. Thus upper bound suggests that the coverage algorithms work especially well when the failure probabilities are small. Reliability problems tend to have small failure probabilities associated with their components. These observations indicate that the coverage algorithms are well suited for solving problems that occur in practice.

The proof of equation (12.1) will be by induction. Let n-tuple $(b_1,\ldots,b_i,*,\ldots,*)$ be a specification of system states where each b_j is either zero or one. Let $D(b_1,\ldots,b_i,*,\ldots,*) = Pr[S \mid (b_1,\ldots,b_i,*,\ldots,*)]$, where S is the set of failure states.

Since $(*,\ldots,*)$ is the set of all system states, $D(*,\ldots,*) = Pr[S]$ which is equal to the denominator of the left-hand side of equation (12.1). Note that

$$D(b_1,\ldots,b_n) = \begin{cases} 1 \text{ if } (b_1,\ldots,b_n) \in S \\ 0 \text{ otherwise} \end{cases}$$

D can be defined inductively as

$$D(b_1,\ldots,b_{i-1},*,\ldots,*) = p_i D(b_1,\ldots,b_{i-1},0,*,\ldots,*) + (1-p_i) D(b_1,\ldots,b_{i-1},1,*,\ldots,*) .$$

If the n-component system has the monotonic property,

$$D(b_1,\ldots,b_{i-1},0,*,\ldots,*) \geq D(b_1,\ldots,b_{i-1},1,*,\ldots,*) .$$

Let

$$N(b_1,\ldots,b_n) = \begin{cases} 1 \text{ if } (b_1,\ldots,b_n) \in S \\ 0 \text{ otherwise} \end{cases} .$$

Define N inductively as

$$N(b_1,\ldots,b_{i-1},*,\ldots,*) = p_i \cdot N(b_1,\ldots,b_{i-1},0,*,\ldots,*) + N(b_1,\ldots,b_{i-1},1,*,\ldots,*) .$$

We claim $N(*,\ldots,*)$ is greater than $\sum_{k=1}^{m} Pr[F_k]$. This can be seen by observing the the contribution of failure state $(b_1,\ldots,b_n)$ to $N(*,\ldots,*)$ is equal to

$$\prod_{i=1}^{n} p_i^{c'_i} \text{ where } c'_i = \begin{cases} 1 \text{ if } b_i = 0 \\ 0 \text{ if } b_i = 1 \end{cases} .$$

Thus, the contribution of all failure states to $N(*,\ldots,*)$ is greater than the contribution of all minimal failure states to $N(*,\ldots,*)$ which is equal to the sum of the probabilities of all the failure sets.

We will show by an induction argument that

$$\frac{N(*,\ldots,*)}{D(*,\ldots,*)} \leq \prod_{i=1}^{n} (1+p_i)$$

which will validate equation (12.1). The induction hypothesis is

$$N(b_1,\ldots,b_i,*,\ldots,*) \leq \prod_{j=i+1}^{n} (1+p_j)\, D(b_1,\ldots,b_i,*,\ldots,*)$$

for all combinations of 0's and 1's substituted for $(b_1,\ldots,b_i)$. The basis of the induction argument is that for all system states $(b_1,\ldots,b_n)$, $N(b_1,\ldots,b_n) = D(b_1,\ldots,b_n)$ by definition of N and D. We will assume the induction hypothesis

for i and show this implies it is true for $i-1$.

$$N(b_1,\ldots,b_{i-1},*,\ldots,*) = p_i \cdot N(b_1,\ldots,b_{i-1},0,*,\ldots,*) + N(b_1,\ldots,b_{i-1},1,*,\ldots,*)$$

$$\leq \prod_{j=i+1}^{n} (1+p_j)[p_i\, D(b_1,\ldots,b_{i-1},0,\ldots,\ldots,*) + D(b_1,\ldots,b_{i-1},1,\ldots,\ldots,*)] =$$

$$\prod_{j=i+1}^{n} (1+p_j)\left[\frac{p_i\, D(b_1,\ldots,b_{i-1},0,*,\ldots,*) + D(b_1,\ldots,b_{i-1},1,*,\ldots,*)}{p_i\, D(b_1,\ldots,b_{i-1},0,*,\ldots,*) + (1-p_i)\, D(b_1,\ldots,b_{i-1},1,*,\ldots,*)}\right] \quad (12.2)$$

$$\cdot\, D(b_1,\ldots,b_{i-1},*,\ldots,*)\,.$$

Formula (12.2) is maximized when $D(b_1,\ldots,b_{i-1},1,*,\ldots,*)$ is maximum, but since $D(b_1,\ldots,b_{i-1},1,*,\ldots,*) \leq D(b_1,\ldots,b_{i-1},0,*,\ldots,*)$, this implies (12.2) is maximized when $D(b_1,\ldots,b_{i-1},1,*,\ldots,*) = D(b_1,\ldots,b_{i-1},0,*,\ldots,*)$. Substituting this value into (12.2) yields the conclusion

$$N(b_1,\ldots,b_{i-1},*,\ldots,*) \leq \prod_{j=i}^{n} (1+p_j) \cdot D(b_1,\ldots,b_{i-1},*,\ldots,*)\,.$$

Thus equation (12.1) is verified.

13. Deterministic Upper and Lower Bounds on $Pr[S]$, An Extension of Boole's Inequality

Boole's inequality states that

$$Pr\left[\bigcup_{k=1}^{m} F_k\right] \leq \sum_{k=1}^{m} Pr[\,F_k\,]. \quad (13.1)$$

We provide a lower bound on $Pr\left[\bigcup_{k=1}^{m} F_k\right]$, and show that this lower bound can be computed quickly. First, we present Boole's inequality in another form. We defined r_i in section 6 as

$$r_i = \sum_{\substack{s \text{ a state s.t.} \\ cov(s)=i}} Pr[s] = Pr[\,cov(s) = i\,]$$

Now we define a positive integer valued random variable Z such that

$Pr[\ Z = i\ ] = r_i$. Let I be the event $Z \geq 1$. Then

$$E[\ Z\ ] = \sum_{i=0}^{m} i \cdot Pr[\ Z = i\ ] = \sum_{i=0}^{m} i \cdot r_i = \sum_{k=1}^{m} Pr[\ F_k\ ]$$

and

$$Pr[\ I\ ] = \sum_{i=1}^{m} Pr[\ Z = i\ ] = \sum_{i=1}^{m} r_i = Pr\left[\bigcup_{i=1}^{m} F_k\right]$$

Thus Boole's inequality can be rewritten as

$$Pr[\ I\ ] \leq E[\ Z\ ]. \tag{13.2}$$

which is a special case of Markov's inequality. Now we develop a lower bound on $Pr[\ I\ ]$.

$$E[\ Z^2\ ] = \sum_{i=1}^{m} i^2 \cdot Pr[\ Z = i\ ] = \sum_{i=1}^{m} i^2 \cdot r_i = \sum_{k=1}^{m} \sum_{j=1}^{m} Pr\left[F_k \cap F_j\right]$$

We claim

THEOREM :

$$\frac{(E[\ Z\ ])^2}{E[\ Z^2\ ]} \leq Pr[\ I\ ] \tag{13.3}$$

or alternatively,

$$\frac{\left[\sum_{k=1}^{m} Pr[\ F_k\ ]\right]^2}{\sum_{k=1}^{m} \sum_{j=1}^{m} Pr\left[F_k \cap F_j\right]} \leq Pr\left[\bigcup_{k=1}^{m} F_k\right] \tag{13.4}$$

Proof :

This is equivalent to proving

$$\frac{\left[\sum_{i=1}^{m} i \cdot r_i\right]^2}{\sum_{i=1}^{m} i^2 \cdot r_i} \leq \sum_{i=1}^{m} r_i$$

which is true if and only if

$$\left[\sum_{i=1}^{m} i \cdot r_i\right] \cdot \left[\sum_{j=1}^{m} j \cdot r_j\right] \leq \left[\sum_{i=1}^{m} r_i\right] \cdot \left[\sum_{j=1}^{m} j^2 \cdot r_j\right]$$

Compare terms $i = l$, $j = k$ and $i = k$, $j = l$ from the left-hand side and right-hand side of the equation.

Left-hand side : $l \cdot r_l \cdot k \cdot r_k + k \cdot r_k \cdot l \cdot r_l = 2 \cdot (k \cdot l) \cdot r_k \cdot r_l$.

Right-hand side : $r_l \cdot k^2 \cdot r_k + r_k \cdot l^2 \cdot r_l = (k^2 + l^2) \cdot r_k \cdot r_l$.

It is sufficient to show $2 \cdot (k \cdot l) \leq k^2 + l^2$. But this follows since $(k - l)^2 \geq 0$. This completes the proof.

Notice that from the list of failure sets we can compute

$$\sum_{k=1}^{m} \sum_{j=1}^{m} Pr\left[F_k \cap F_j\right]$$

in time $O(m^2 \cdot n)$. We now consider how good these bounds are by taking the ratio of the upper bound and the lower bound. This ratio is equal to

$$\frac{E[Z^2]}{E[Z]} = \frac{\sum_{i=1}^{m} i^2 \cdot r_i}{\sum_{i=1}^{m} i \cdot r_i}$$

The best a priori upper bound on this ratio is m, but in practice we expect this ratio to be much smaller than m.

Inequality (13.3) can be rewritten as $\frac{E[Z^2]}{E[Z]} \geq \frac{E[Z]}{Pr[I]} = \frac{\sum_{i=1}^{m} Pr[F_i]}{Pr[S]}$.

Thus for the algorithms presented in Sections 7 and 11, $\frac{E[Z^2]}{E[Z] \cdot \delta \cdot \varepsilon^2}$ is an easily computable upper bound on the number of trials sufficient to guarantee an (ε, δ) algorithm.

14. A Computational Example

The new coverage algorithms was applied to the network reliability problem used as an example in Section 2. The failure probability of the system is .21254, there are nine failure sets,

$$\sum_{k=1}^{9} Pr[F_k] = .2644$$

and

$$\frac{\left[\sum_{k=1}^{9} Pr[\,F_k\,]\right]^2}{\sum_{k=1}^{9}\sum_{j=1}^{9} Pr\left[F_k \cap F_j\right]} = \frac{.2644^2}{.3953} = .1768$$

Four different versions of the coverage algorithm are used to estimate $Pr[S]$. Since randomly selecting block (s, k) in the first two steps of a trial is the same for all versions of the algorithm, all four versions use the same randomly selected block (s, k) on the same trial.

The four different versions of the coverage algorithm differ only in the computation of $\alpha(s, k)$. The method used to compute $\alpha(s, k)$ for each of the four versions is described in the following table.

Version	Computation of $\alpha(s, k)$
1	$\alpha(s, k) = \frac{1}{cov(s)}$
2	$i = \min_{j=1,\ldots,m} \{j \mid s \in F_j\}$ $\alpha(s, k) = \begin{cases} 1 \text{ if } i = k \\ 0 \text{ if } i \neq k \end{cases}$
3	randomly select F_i with probability $\frac{1}{m}$ until $s \in F_i$. Let l = number of failure sets picked until $s \in F_i$ $\alpha(s, k) = \frac{l}{m}$
4	let F_i be failure set s.t. $s \in F_i$ found using version 3 $\alpha(s, k) = \begin{cases} 1 \text{ if } i = k \\ 0 \text{ if } i \neq k \end{cases}$

Forty trials were conducted, with the following results. When Version 1 was used, the estimate of $Pr[S]$ determined by these forty trials is .2181. The estimates derived by treating each of the first, second, third and fourth sets of ten trials as though it were the entire sample are .2005, .2291, .2181 and .2247, respectively.

When Version 2 was used, the estimate of $Pr[S]$ determined by these forty trials is .2247. The estimates derived by treating each of the first, second, third and fourth sets of ten trials as though it were the entire sample are .2115, .2115, .2300 and .2380, respectively.

When Version 3 was used, the estimate of $Pr[S]$ determined by these forty trials is .2078. The estimates derived by treating each of the first, second, third and fourth sets of ten trials as though it were the entire sample are .1410, .2262, .1410 and .3232, respectively.

When Version 4 was used, the estimate of $Pr[S]$ determined by these forty trials is .2115. The estimates derived by treating each of the first, second, third and fourth sets of ten trials as though it were the entire sample are .2380, .2380,

.1851 and .1851, respectively.

A detailed table of these results follows.

k	system state $s \in F_k$	failure sets containing s	$\alpha(s, k)$ version 1	2	3	4	selected index of F_i s.t. $s \in F_i$
6	00100010	3,6,9	1/3	0	2/9	1	6
5	10000111	5,8	1/2	1	7/9	1	5
5	00000011	5,6,8,9	1/4	1	1/9	1	5
4	11011010	4	1	1	11/9	1	4
9	00001011	8,9	1/2	0	2/9	0	8
5	11010111	5	1	1	1/9	1	5
9	00111010	9	1	1	2/9	1	9
9	00011011	9	1	1	5/9	1	9
9	00111011	9	1	1	4/9	1	9
9	00111111	9	1	1	13/9	1	9
3	01101010	3	1	1	14/9	1	3
9	00100000	1,2,3,6,7,9	1/6	0	1/9	0	3
9	00110011	9	1	1	1/9	1	9
5	11010111	5	1	1	2/9	1	5
1	10011100	1	1	1	15/9	1	1
9	00110111	9	1	1	4/9	1	9
5	10010001	2,5	1/2	0	2/9	1	5
9	00111011	9	1	1	3/9	1	9
5	11000011	5	1	1	8/9	1	5
5	10010011	5	1	1	18/9	1	5
2	10000001	2,5,7,8	1/4	1	7/9	0	7
8	10000011	5,8	1/2	0	5/9	0	5
5	11000011	5	1	1	2/9	1	5
9	00011111	9	1	1	10/9	1	9
5	11010101	5	1	1	1/9	1	5
8	10001111	8	1	1	14/9	1	8
4	11000010	4,5	1/2	1	1/9	0	5
8	10001011	8	1	1	1/9	1	8
5	10010011	5	1	1	2/9	1	5
9	00111111	9	1	1	5/9	1	9
8	10001011	8	1	1	4/9	1	8
9	00101011	9	1	1	14/9	1	9
5	11000011	5	1	1	5/9	1	5
4	10011010	4	1	1	24/9	1	4
8	10000110	5,8	1/2	0	3/9	0	5
5	10000111	5,8	1/2	1	6/9	0	8
5	11010111	5	1	1	12/9	1	5
7	10001001	7,8	1/2	1	8/9	0	8
8	10001111	8	1	1	4/9	1	8
9	00011111	9	1	1	30/9	1	9

15. Conclusion

We have presented several highly effective Monte-Carlo methods for estimating the failure probability of an n-component system, given a list of failure sets. In many practical situations an n-component system is presented as a network or fault tree, and the failure sets are too numerous to be explicitly listed. In future work we will show that our coverage formula and the associated Monte Carlo method can sometimes be applied in such situations using implicit methods of sampling from among the failure sets of the system.

16. Acknowledgements

The authors would like to thank J. Pitman for pointing out the approach to proving that the Linear Time Coverage Algorithm is an (ε, δ) algorithm using Kolmogorov's Inequality.

17. Nomenclature

The nomenclature introduced in each section is listed here for easy reference.

Section 1 n - number of components in the system

m - number of failure sets

p_i - probability component i is failing

$(b_1,\ldots,b_n)$ - a specification of a system state

$b_i = 0$ if component i is failing

$b_i = 1$ if component i is working

$(c_1,\ldots,c_n)$ - a failure set specification

$c_i = 0$ if component i is failing

$c_i = 1$ if component i is working

$c_i = *$ if component i may be either failing or working

F - a failure set

F_k - the k^{th} failure set

s - a system state

S - the set of all failure states

$Pr[S]$ - the probability the network is in a failing state

Section 2 x, y - the two designated nodes in the two terminal problem

Section 3 E - enclosing region

$A(E)$ - area of region E

U - unknown region

$A(U)$ - area of region U

b - number of blocks into which E is subdivided

a_i - area of block i in E

α_i - 1 if block $i \in U$, 0 if block $i \notin U$.

Y - random variable s.t. $E[Y] = A(U)$

Section 4 N - number of trials performed during the course of a Monte-Carlo algorithm

Y_i - estimator produced by the i^{th} trial

μ - $E[Y_i]$

σ^2 - variance of Y_i

ε - allowable relative error

δ - confidence level that the relative error is $\leq \varepsilon$

(ε, δ) - algorithm with confidence level δ the relative error of the estimator produced by the algorithm is less than or equal to ε.

Section 6 $cov(s)$ - the number of failure sets F_k s.t. $s \in F_k$

(s, k) - a block in the region E for the coverage algorithms $S \in F_k$

$\alpha(s, k)$ - indicates whether or not $(s, k) \in U$

$\alpha(s\ k) = 1$ if $(s, k) \in U$

$\alpha(s\ k) = 0$ if $(s, k) \notin U$

Section 7 FS - an array of size m used to pick failure sets

r_i - summation of the probability of all states with coverage i

Section 8 $\alpha(s, k)$ - generalized version of definition given in Section 6

$$\sum_{\{k \mid s \in F_k\}} E[\alpha(s, k)] = 1$$

Section 9 c - the cutoff

Section 10 Y - unbiased estimator of $Pr[S]$

Y' - a second unbiased estimator of $Pr[S]$

$\alpha(s, k)$ - used to produce estimator Y

$\alpha'(s, k)$ - used to produce estimator Y'

$X(s, k)$ - length of trial given that block (s, k) is selected

ϑ - $\dfrac{E[Y^2]}{E[Y]^2}$

ϑ' - $\dfrac{E[Y'^2]}{E[Y']^2}$

μ - $\dfrac{Pr[S]}{\sum_{k=1}^{m} Pr[F_k]}$

t - number of times (*) in Step 3 of algorithm is executed during the course of the algorithm.

X_j - length of the j^{th} trial

S_n - $S_n = \sum_{i=1}^{n} X_i$

Y_j - value of Y on the j^{th} trial

Y_j' - value of Y' on the j^{th} trial

R_n - $\sum_{i=1}^{n} Y_i$

R_n' - $\sum_{i=1}^{n} Y_i'$

$N(t)$ - number of trials completed after (*) in Step 3 is executed t times

$\bar{c}$ - both ϑ and ϑ' are $\leq \frac{\bar{c}}{u}$

18. Linear Time Coverage Algorithm Pascal Program

In the following pages there is a listing of a Pascal program for the Linear Time Coverage Algorithm described in Section 10. After the problem data is input and the preprocessing is performed, the upper bound and the lower bound on $Pr[\,S\,]$ described in Section 13 is computed and output. Following this are steps 1 thru 6 of the Linear Time Coverage Algorithm. Following this listing is a run of the program using as input data the example problem described in Section 2.

```
program montel (input,output) ;

{ this program computes the failure probability of an n-component system
  where the input is a list of failure sets -
  the running time of this algorithm is c * numcomponents * numfail
  divided by delta * epsilon * epsilon, where
  c is a small constant  - (see the proof that this algorithm is an
                            epsilon,delta algorithm)
  numcomponents is  the number of system components
  numfail is the number of specified failure sets
  delta is the confidence level
  epsilon is the allowable relative error }

  label 1 ;
  var i,j,k,l : integer ;
      numcomponents : integer ;
      numfail : integer ;
      Seed : integer ;
      prob : array [1..40] of real ;
      failset : array [1..100,1..40] of integer ;
      failprob : array [0..100] of real ;
      sumprob : real ;
      sstate : array [1..40] of integer ;
      numsteps : integer ;
      sumest1 : real ;
      sumest2 : real ;
      outest1 : real ;
      outest2 : real ;
      numtrials : integer ;
      c : real ;
      x : real ;
      z : real ;
      epsilon : real ;
      delta : real ;
      time : integer ;
      alpha1 : real ;
      alpha2 : real ;
      found : boolean ;

 {select failure set }

 function selectfail : integer ;
 var low,high,pnt1,pnth : integer ;
     x : real ;
     found : boolean ;
 begin
    x := random (Seed) ;
    found := false ;
    low := 0 ;
    high := numfail ;
    while (not found) do
    begin
       pnt1 := (low + high) div 2 ;
       pnth := pnt1 + 1 ;
       if (failprob[pnt1] >= x) then high := pnt1 ;
       if (failprob[pnth] < x) then low := pnth ;
       if ((failprob[pnt1] <=x) and (failprob[pnth] >= x)) then found := true
    end ;
    selectfail := pnth
```

```
end ;

{ select system state }

procedure selectstate ;
   var i : integer ;
       x : real ;
   begin
        for i := 1 to numcomponents do
        begin
            sstate[i] := failset[k,i] ;
            if (sstate[i] = -1) then
            begin
                x := random (Seed) ;
                if (x <= prob[i]) then sstate[i] := 0
                else sstate[i] := 1
            end
        end
   end ;

{ function to see if system state is in failure set }

function inset : boolean ;
var indic : boolean ;
    j : integer ;
begin
    indic := true ;
    for j := 1 to numcomponents do
    begin
       if ((failset[i,j] = 1) and (sstate[j] = 0)) then
          indic := false ;
       if ((failset[i,j] = 0) and (sstate[j] = 1)) then
          indic := false
    end ;
    inset := indic
end ;

{ input problem data }

   begin
      writeln ('Seed:') ;
      read (Seed) ;
      i := seed (Seed) ;
      writeln ('enter epsilon :') ;
      read (epsilon) ;
      writeln ('enter delta :') ;
      read (delta) ;
      writeln ('enter constant :') ;
      read (c) ;
      writeln ('number of components :') ;
      read (numcomponents) ;
      writeln ('input probability of components : ') ;
      for i := 1 to numcomponents do
      begin
          writeln ('input prob. of component ',i:3) ;
          read (prob[i])
      end ;
      writeln ('input number  of failure sets :') ;
      read (numfail) ;
```

```
    writeln ;
    writeln ('The failure set specification format is :') ;
    writeln ('1 - component must work for system state to be in failure set') ;
    writeln ('0 - component must fail for system state to be in failure set') ;
    writeln ('-1 - component may either work or fail (unspecified)') ;
    writeln ;
    for i := 1 to numfail do
    begin
       writeln ('input specifications for failure set ',i:3) ;
       for j := 1 to numcomponents do
          read (failset[i,j])
    end ;

{ preprocessing }

    failprob[0] := 0.0 ;
    sumprob := 0 ;
    for i := 1 to numfail do
    begin
        failprob[i] := 1.0 ;
        for j := 1 to numcomponents do
        begin
            if (failset[i,j] = 0) then
                failprob[i] := failprob[i] * prob[j] ;
            if (failset[i,j] = 1) then
                failprob[i] := failprob[i] * (1.0 - prob[j])
        end ;
        sumprob := sumprob + failprob[i]
    end ;
    writeln ;

{ Print upper bound on the failure probability }

    writeln ('Upper bound on the failure probability is ',sumprob:12:8) ;

{ Compute and print lower bound on the failure probability }

    z := 0.0 ;
    for i := 1 to numfail do
        for j := 1 to numfail do
        begin
            x := 1.0 ;
            for k := 1 to numcomponents  do
            begin
                if ((failset[i,k]=1) or (failset[j,k]=1)) then
                    x := x * (1.0 - prob[k]) ;
                if ((failset[i,k]=0) or (failset[j,k]=0)) then
                    x := x * prob[k] ;
                if ((failset[i,k] <> failset[j,k]) and (failset[i,k] <> -1)
                    and (failset[j,k] <> -1)) then x := 0
            end ;
            z := z + x
        end ;
    writeln ; writeln (' E(Z*Z) = ',z:12:8) ;  writeln ;
    z := (sumprob * sumprob) / z ;
    writeln ;
    writeln ('Lower bound on the failure probability is ',z:12:8) ;
    writeln ;
```

```
      x := 0 ;
      for i := 1 to numfail do
      begin
          writeln ('The prob. of failure set ',i:3,' is ',failprob[i]:12:8) ;
          x := failprob[i] + x ;
          failprob[i] := x / sumprob
      end ;
      numsteps := trunc ((c * numfail) / (delta * epsilon * epsilon)) + 1 ;
      writeln ; writeln ('the number of steps will be ',numsteps:6) ;
      time := 0 ;
      numtrials := 0 ;
      sumest1 := 0.0 ;
      sumest2 := 0.0 ;

{ beginning of Monte-Carlo trial }

      while (time <= numsteps) do
      begin

{ step 1 - select failure set }

          k := selectfail ;

{ step 2 - select system state }

          selectstate ;

{ step 3 - compute estimators alphal and alpha2 }

          l := 0 ;
          found := false ;
          while (not found) do
          begin
              i := trunc (random(Seed) * numfail) + 1 ;
              if (i >= numfail) then i := numfail ;
              l := l + 1 ;
              time := time + 1 ;
              if (time > numsteps) then goto 1 ;
              found := inset
          end ;

{ step 4 - compute alphal and alpha2 }

          alphal := (1 / numfail) * sumprob ;
          if (k = i) then alpha2 := sumprob
          else alpha2 := 0 ;

{ step 5 - increment number of trials and total estimators }

          numtrials := numtrials + 1 ;
          sumest1 := sumest1 + alphal ;
          sumest2 := sumest2 + alpha2
      end  { while (time <= numsteps) } ;

{ step 6 - the simulation is completed, output the results }

  1 : outest1 := sumest1 / numtrials ;
      outest2 := sumest2 / numtrials ;
      writeln ('number of trials completed = ',numtrials:5) ;
```

```
    writeln ('estimator 1 = ',outest1:12:8) ;
    writeln ('estimator 2 = ',outest2:12:8)
end  { of program } .
```

```
Seed:
39845
enter epsilon :
0.1
enter delta :
0.05
enter constant :
8.0
number of components :
8
input probability of components :
input prob. of component   1
0.1
input prob. of component   2
0.5
input prob. of component   3
0.4
input prob. of component   4
0.3
input prob. of component   5
0.2
input prob. of component   6
0.4
input prob. of component   7
0.1
input prob. of component   8
0.2
input number  of failure sets :
9

The failure set specification format is :
1 - component must work for system state to be in failure set
0 - component must fail for system state to be in failure set
-1 - component may either work or fail (unspecified)

input specifications for failure set   1
-1 -1 -1 -1 -1 -1  0  0
input specifications for failure set   2
-1 -1 -1 -1  0  0  0 -1
input specifications for failure set   3
 0 -1 -1  0 -1  0 -1  0
input specifications for failure set   4
-1 -1  0 -1 -1  0 -1  0
input specifications for failure set   5
-1 -1  0 -1  0 -1 -1 -1
input specifications for failure set   6
 0 -1 -1  0  0 -1 -1 -1
input specifications for failure set   7
-1  0 -1  0 -1  0  0 -1
input specifications for failure set   8
-1  0  0  0 -1 -1 -1 -1
input specifications for failure set   9
 0  0 -1 -1 -1 -1 -1 -1

Upper bound on the failure probability is   0.26440000

 E(Z*Z) =   0.39534400
```

```
Lower bound on the failure probability is   0.17682666

The prob. of failure set   1 is   0.02000000
The prob. of failure set   2 is   0.00800000
The prob. of failure set   3 is   0.00240000
The prob. of failure set   4 is   0.03200000
The prob. of failure set   5 is   0.08000000
The prob. of failure set   6 is   0.00600000
The prob. of failure set   7 is   0.00600000
The prob. of failure set   8 is   0.06000000
The prob. of failure set   9 is   0.05000000

the number of steps will be 144001
number of trials completed = 19854
estimator 1 =   0.21307693
estimator 2 =   0.21215656
```

19. References

[1] J. Scott Provan and Michael O. Ball, *The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected*, working paper MS/S 81-002, Management Science and Statistics, January 1981 (revised April 1981)

[2] S. Tsukiyama, I. Shirakawa, H. Ozaki, H. Ariyoshi, *An Algorithm to Enumerate All Cutsets of a Graph in Linear Time*, JACM, vol. 27, no. 4, October 1980, pp. 619-632

[3] S. Ross, *Applied Probability Models with Optimization Applications*, 1970, chapter 3, section 3.4, p. 37

[4] W. Feller, *An Introduction to Probability Theory and Its Applications*, third edition, vol. 1, 1968, chapter 9, section 7, pp. 234-235