Optimal Design of Checks for Error Detection and Location in Fault-Tolerant Multiprocessor Systems

Ramesh K. Sitaraman and Niraj K. Jha, *Senior Member, IEEE*

Abstract— **Designing checks to detect or locate errors in the data plays an important role in the design of fault tolerant systems. Recently, the problem of synthesizing the data-check (DC) relationship has received a lot of attention in the context of a natural paradigm for concurrent error detection/location known as *algorithm-based fault tolerance* (ABFT). Banerjee and Abraham have shown that an ABFT scheme can be modeled as a tripartite graph consisting of processors (P), data (D), and checks (C). Any technique for designing ABFT systems requires a procedure for synthesizing a DC relationship, which not only has a low overhead but also has all the properties required by the designer.**

The main contribution of this work is to propose a simple and novel algorithm called RANDGEN to generate DC graphs. This synthesis approach itself is very fast and can be fully parallelized. By simply varying its parameters, the same algorithm RANDGEN can produce DC graphs with a wide spectrum of properties, many of which have been considered very important in recent ABFT designs.

RANDGEN produces s-error-detectable DC graphs with asymptotically the least number of checks for the first time. RANDGEN can also produce s-error-locatable DC graphs using only a small number of checks. This is the first general procedure for producing error-locatable graphs for any value of s. Another important outstanding problem in DC graph design is providing fast and practical methods for actually locating the errors in the data from the output pattern at the checks. We show that RANDGEN can be used to design DC graphs, which permit easy diagnosis, again with a small number of checks. It has been pointed out previously that "uniform" checks may simplify the design of the ABFT system. We show how RANDGEN can be modified very simply to produce uniform s-error-detectable/locatable DC graphs. Finally, we show how one can generalize these results to synthesize *strictly* s-error-detectable/locatable DC graphs which can detect/locate up to s data errors even when s or fewer check computations are erroneous.

Index Terms— **Algorithm-based fault tolerance, checksum encoding, concurrent error detection, concurrent error location, majority diagnosis, randomized algorithms, uniform checks.**

Manuscript received July 1, 1991; revised January 24, 1992, September 5, 1992. This work was based on "Optimal Design of Checks for Error Detection and Location in Fault Tolerant Multiprocessor Systems" by R. K. Sitaraman and N. K. Jha, which appeared in the *Proceedings of the 5th International Conference on Fault Tolerant Computing Systems,* Nurenberg, Germany, September 1991, pp. 396–406. © 1991 Springer-Verlag. This work was supported by DARPA/ONR under Contract N00014-88-K-0459, the NSF under Grant CCR-9057486, a grant from MITL, ONR under Contract N00014-91-J-1199, and AFOSR under Contract AFOSR-90-0144.

R. Sitaraman is with the Department of Computer Science, Princeton University, Princeton, NJ 08544.

N. K. Jha is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544.

IEEE Log Number 9208477.

I. Introduction

THE QUESTION OF how to use a *minimum* number of checks for data such that one can locate (or just detect) any set of at most s errors is an important question in fault tolerance. Much recent interest in designing data-check relationships for error detection and location has been in the area of algorithm-based fault tolerance (ABFT). ABFT was introduced as a technique to detect and locate errors in matrix computations [13]. There have been many applications of this technique to a variety of problems, including fast Fourier transforms [8], [15], [25], [30], sorting [9], and signal processing applications such as matrix multiplication, matrix inversion, LU decomposition, QR decomposition, FIR filtering, etc. [7], [13], [14], [16], [18], [23], [27]. It has also been applied to various architectures such as linear array [1], [14], mesh array [13] and hypercube [3]. ABFT is a very attractive method for concurrent error detection and fault location due to its low hardware and time overhead. Many methods for analyzing ABFT systems also exist [4], [12], [17], [19], [24], [26].

In [4], a graph-theoretic model for studying ABFT schemes was proposed. The scheme was represented as a tripartite graph whose vertex set was $P \cup D \cup C$ and its edge set was $PD \cup DC$, where P, D, and C are the set of processors, data, and checks, respectively, and PD and DC are the edges between P and D and between D and C, respectively. An edge $(u, v) \in PD$ implies that processor u affects the value of data element v in the computation; that is, if processor u fails, v could have an error. An edge $(v, z) \in DC$ implies that check z checks data element v. The set of data elements affected by a processor $u \in P$ is said to be its *data set.* The set of data elements checked by a check $z \in C$ is said to be its *error set* . In this paper, the checks are assumed to be of the simplest type. A check operates on any nonempty subset of the data and will detect exactly one error in its error set. More formally, a check z outputs a binary value of 0 or 1. It is 0 when all the data elements in its error set are error-free. It is 1 when *exactly* one of the data elements is in error. If there is more than one error in its error set, its output value is arbitrary and, hence, the check is undependable.

The graph model is independent of the exact implementation of the check. However, one simple implementation of such a check is to use a (unweighted) checksum of all the data elements in the error set. The checks themselves are computed by some dedicated processors. For example, if our checks are simple checksums, each such processor will independently compute the value of its assigned output checksum from the system inputs and then compare it with the checksum obtained

from the data elements in its error set. It is important to note that *processors dedicated to computing checks are not represented in the PDC graph in this approach.* For the sake of simplicity of presentation, first, we assume that either the processors computing the checks are not faulty or they are capable of exposing their own faults (by employing any concurrent error detection technique). This assumption and the preceding checking philosophy are also inherently present in all previous papers on PDC graphs-based ABFT system design [4], [12], [20], [21], [28]. However, in Section VII, we show how we can easily extend the results obtained in the previous sections of our paper to the more realistic situation when these processors dedicated to checking can also become faulty and have no alternate way of exposing their own faults.

One of the main goals of research in ABFT is to design efficient systems that are t-fault locatable (or detectable), that is, assuming that not more than t processors can fail in a computation one would like to locate exactly which, if any, of the processors failed (or simply detect that there has been a failure). As is traditionally done, here we assume that a faulty processor results in an error in at least one of its data elements in the computation. The importance of the graph-theoretic model is that the fault detectability and locatability properties of the computation can be derived directly as a property of the tripartite graph.

Designing t-fault-detectable or -locatable systems involves many degrees of freedom. One could assume that the architecture is not chosen *a priori*. In this case one could add checks to the algorithm to make it error tolerant and then project its data dependence graph to obtain the best fault tolerant architecture [27]. This could be said to be a synthesis for fault tolerance approach. Or, alternately, one could assume that the algorithm and architecture are already given, that is, the PD graph is fixed, and that the checks must be added for some desired fault tolerance [4], [12], [20], [21], [28]. This could be called a design for fault tolerance approach. Let the maximum number of data elements affected by any t processors in the given PD graph be MAX. One of the known approaches for designing t-fault-detectable/locatable PDC graphs is to design a DC graph that is s-error detectable/locatable such that s equals MAX [4]. The unit system approach [20], [21], [28], on the other hand, first generates a unit system in which each element of P is connected to exactly one element of D. For this unit system, a t-error-detectable/locatable DC graph is synthesized. The PDC graph formed by either taking the product of the given nonfault tolerant system with the unit system [20], [21] or by taking the composite of various unit systems [28] gives us a t-fault-detectable/locatable system. For the purpose of this paper, it is important to note that in any of these methodologies we require a systematic procedure to design DC relationships that can detect or locate a specified number of errors. We now formally define what it means for a DC graph to detect/locate s errors.

Lemma 1.1: A DC graph is s-error detectable iff every possible nonempty set of errors in the data of cardinality at most s makes at least one of the checks output a 1.

Lemma 1.2: A DC graph is s-error locatable iff every possible set of errors in the data of cardinality at most s gives a different output pattern at the checks, that is, no two distinct sets of errors of cardinalities at most s can give the same output pattern at the checks.

In Section II, we describe a versatile algorithm that can be used to generate DC graphs with a variety of properties. Section III deals with error-detectable DC graphs. In Section IV, we consider error-locatable graphs. In Section V, we show how to use this algorithm to design DC graphs for easy diagnosis. Section VI deals with a generation of "uniform" checks. Section VII considers the case when the processors computing the checks can also fail. This section shows that our results extend naturally to deal with erroneous check computations as well. In the last section, we provide some concluding remarks.

II. Algorithm for Generating DC Graphs

In this section, we propose a simple, efficient, and versatile algorithm called RANDGEN (RANDom GENeration). By just varying the input parameters of RANDGEN one can synthesize, using only a small number of checks, DC graphs with a wide range of properties that researchers in ABFT have found to be useful and important in their designs.

Let D be the set of data elements and C be the set of checks. Let the number of data elements ($|D|$) be n. The number of checks ($|C|$) is denoted by c. The DC graph can be obtained by constructing tuples (called edges) (u, v), where u is a data element and v is a check. The construction algorithm RANDGEN is novel in that it is probabilistic, that is, it makes random decisions during the course of the construction by perhaps using a random number generator. To construct a DC graph with n data elements, algorithm RANDGEN takes two arguments: the number of checks, c, that can be used and a probability p. The algorithm is as follows:

Algorithm RANDGEN(c, p)
Let D be the set of n data elements and C be the set of c checks.
For every pair (u, v), where $u \in D$ and $v \in C$ do:
Add edge (u, v) to the DC graph with probability p.
End RANDGEN

It is easy to see that RANDGEN is very fast and simple to implement. It can also be easily executed in parallel since essentially the decision to include or not include an edge is taken independently of other edges. We state this formally as a theorem.

Theorem 2.1: Algorithm RANDGEN runs in time $O(c.n)$, where c is the number of checks in the graph and n the number of data elements. It can also be executed in a constant number of steps with $c.n$ processors.

Proof: It is easy to see that RANDGEN takes constant time for each possible edge (u, v), where u is a data element and v is a check. Hence the total time taken is $O(c.n)$. That each pair (u, v) can be considered in parallel is also clear. □

It must be mentioned that since RANDGEN is probabilistic, our results will show that with an overwhelmingly large

probability, the DC graphs produced will have the required properties.[1]

The minimum number of checks required for a DC graph with n data elements to be s-error-detectable has been shown to be $\Omega(s \log n)$ [12]. We show how to construct an asymptotically optimal s-error-detectable DC graph, that is, using only $O(s \log n)$ checks. Previously, there were no general methods known for designing s-error-locatable graphs. A (loose?) lower bound for minimum number of checks required for a DC graph to be s-error locating is $\Omega(s \log n)$ [4]. RANDGEN produces DC graphs with $O(s^2 \log n)$ number of checks which is very close to asymptotically optimal since typically $s \ll n$.

It must be noted that s-error locatability only ensures that no two distinct sets of error patterns of size s or less have the same output pattern at the checks. It does not provide us with an efficient algorithm to diagnose, that is, actually locate the errors from the output pattern at the checks. No efficient algorithm for diagnosis is currently known for a general s-error-locatable DC graph. In fact, the known algorithm is enumerative in its approach and tries to enumerate various possible sets of errors and check if the output pattern at the checks can be caused by them [26] and could be time-consuming for even small values of s. As it is not clear that this algorithm can be improved upon drastically, we need to *design* DC graphs explicitly for easy diagnosis. We introduce a class of s-error-locatable DC graphs which allow easy diagnosis and show how RANDGEN can generate them with only a constant factor overhead in the number of checks. "Uniform" checks [28], which are all identical and check the same number of data elements, have been shown to simplify ABFT design. We show how RANDGEN can be modified to produce such uniform checks for s-error detection/location. Finally, we show how RANDGEN can produce DC graphs that can detect/locate errors in data even when the checks themselves can fail.

III. Error Detectability

We start with the problem of error detection. The conditions of Lemma 1.1 imply a minimum number of checks that one necessarily requires for a DC graph to be s-error detecting. The following lower bound from [12] is stated without proof.

Theorem 3.1: The number of checks, c, for s-error detectability is $\Omega(s \log n)$.[2]

The main theorem of this section shows that RANDGEN produces (with high probability) an s-error-detectable DC graph when parameter $c = 3.8s \log n$ and parameter $p = 1/s$ are used. From the previous lower bound result, it can be seen that this algorithm requires asymptotically the least number of checks. Before we prove the theorem, we illustrate the algorithm RANDGEN for a typical problem. In [5], the problem of encoding a matrix of dimension 1024×1024 and analyzing the reliability of various matrix multiplication algorithms is considered. We will use the encoding of 1024×1024 data values for s-error detection and location as a running example to illustrate our constructions.

Example 3.1: Suppose we would like to construct a DC graph that can detect up to three $(= s)$ errors for a data set consisting of a matrix of dimension 1024×1024, that is, 1 048 576 data elements. We take 228 checks and with each check we do the following. We consider every data element and include a data element in the error set of this check with a probability of one-third $(= 1/s)$. When we are done with this process, we are left with a DC graph that is 3-error detecting with a probability of at least $1 - (1/1\,048\,575)$, very close to 1. As a basis for comparison, notice that the traditional matrix row and column checksum method, which can detect up to three errors, requires 2047 checks.

It should be pointed out that efficient methods have already been given in [12] specifically for the particular cases of $s = 2, 3$, and 4. Then they give a special method for detecting up to seven errors. However, as a comparison, for this example that method would require 570 checks for detecting five, six, or seven errors, whereas our method would require 380, 456, and 532 checks, respectively. For $s > 7$ they have given a general construction method. For s ranging from 8 to 15 they would require 9120 checks, whereas our method would require only 608, 684, 760, 836, 912, 988, 1064, and 1140 checks, respectively. As the value of s and/or n increases, our method performs relatively even better than the method in [12]. One must note, however, that their method is deterministic whereas ours is probabilistic.

Before we prove the main theorem of this section, we state without proof this simple lemma, which we will use repeatedly in this paper.

Lemma 3.1: Given a series of events $E_1, E_2, \cdots, E_k$, $Prob(\cup_i E_i) \leq \Sigma_i Prob(E_i)$.

Theorem 3.2: The algorithm RANDGEN, using parameter $c = 3.8s \log n$ and $p = 1/s$, produces an s-error-detectable DC graph with probability at least $1 - [1/(n-1)]$. The time complexity of constructing this graph is only $O(sn \log n)$.

Proof: The algorithm RANDGEN clearly works for $s = 1$, since $p = 1$ and every check is connected to all data elements. Of course, one such check would suffice. So we will assume that $s > 1$. We need to show that the DC graph satisfies the conditions of Lemma 1.1, that is, every nonempty set $S \subset D, |S| \leq s$, has a check z such that it is connected to exactly one element of S. Let E_S represent the event that there exists no such check for some set S. The probability that the DC graph is not s-error detectable is simply the probability of $\cup_S E_S$, where S takes on the value of all nonempty subsets of D with cardinality not more than s. We will split this union of events into smaller unions as follows and bound each separately. Throughout this paper, e represents the transcendental number 2.718 281 8 . . . .

Let event $A_i, 1 \leq i \leq s$, be $\cup_S E_S$, where S takes all subsets of D of cardinality i. For any single set $S, |S| = i$, and a particular check z, the probability that z is not connected to exactly one element of S, that is, that this check is "bad," is clearly $1 - ip(1-p)^{i-1}$. We now choose $p = 1/s$, which minimizes this expression for $i = s$. Observe that, for this

[1] The proof techniques used in this paper are reminiscent of the probabilistic method in combinatorics pioneered by Erdos and Spencer [10]. Interested readers may also refer to books on the theory of random graphs [2].

[2] All logarithms in this paper are to base 2.

value of p,

$$1 - ip(1-p)^{i-1} = 1 - i\frac{1}{s}\left(1-\frac{1}{s}\right)^{i-1} \leq 1 - i\frac{1}{s}\left(1-\frac{1}{s}\right)^{s-1} \leq 1 - i\frac{1}{se}. \quad (1)$$

From the independence in choosing the edges, the probability that all checks are bad is $(1 - ip(1-p)^{i-1})^c$. We next bound the probability of event A_i.

$$\begin{aligned} Prob(A_i) &\leq \sum_{S,|S|=i} Prob(E_S) \\ &\leq n^i(1 - ip(1-p)^{i-1})^c \\ &\leq n^i e^{-cip(1-p)^{i-1}} \\ &= n^i e^{-ci(1/s)[1-(1/s)]^{i-1}}, \quad \text{for } p = \frac{1}{s} \\ &\leq n^i e^{-[(2e/\log e)s \log n]i(1/se)} \\ &= n^i e^{-2(\log n/\log e)i} \\ &= n^i (e^{1/\log e})^{-2i \log n} \\ &= n^i 2^{-\log n^{2i}}, \quad \text{since } e^{1/\log e} = 2 \\ &= \frac{n^i}{n^{2i}} \\ &= \frac{1}{n^i} \end{aligned} \quad (2)$$

using Lemma 3.1, then using the fact that $1 - x \leq e^{-x}$ and finally using (1) and choosing $c = 3.8s \log n \geq (2e/\log e)s \log n$.

Now, using Lemma 3.1, the probability of the DC graph being bad, that is, not satisfying the conditions of Lemma 1.1, is simply

$$\begin{aligned} &Prob(\cup_{1\leq i\leq s} A_i) \\ &\quad \leq \sum_{1\leq i\leq s} Prob(A_i) \leq \frac{1}{n} + \frac{1}{n^2} + \cdots + \frac{1}{n^s} \\ &\quad \leq \frac{1}{n-1}. \end{aligned}$$

Thus the probability of a "good" DC graph, that is, one that does satisfy the conditions of Lemma 3.1, is at least $1 - [1/(n-1)]$. RANDGEN's time complexity follows from Theorem 2.1. □

For some applications, one may want to decrease even further the probability that the constructed DC graph is not s-error detectable at the cost of adding more checks. One can decrease this probability very quickly by the addition of some extra checks. In our example, we can add 114 more checks to make the total number of checks 342, and the probability of a bad DC graph goes down rapidly to $1/(1\,048\,576^2 - 1) \approx 1/10^{12}$. We can decrease this again by adding more checks if need be.

Corollary 3.1: The algorithm RANDGEN, using $c = (3.8s + 1.9sk) \log n$ checks and $p = 1/s$, produces an s-error-detectable DC graph with probability at least $1 - [1/(n^{k+1} - 1)]$.

Proof: Follows from the proof of the previous theorem by substituting the new value for c at the appropriate step. □

From the preceding discussions it is clear that there is a close relationship between the probability of getting a good DC graph and the number of checks c. After fixing this probability to a value that one will be satisfied with, one can do the computation backward and find the corresponding value of c.

One possible criticism of the preceding approach is that the probability of getting a good DC graph cannot be made 1, although it can be made arbitrarily close to 1. However, one should remember that any fault tolerant design has an *inherent* chance of failure. A system that is assured to catch s faults/errors will fail in the unlikely (but still probable) event that more than s faults/errors occur. As long as the probability of getting a bad DC graph is small compared with the other reasons for failure such as presented earlier, there should be no cause to worry. Even so, the degradation in this construction is "gradual." Even a bad DC graph, improbable as it may be, will still detect most sets of s or fewer errors.

If the designer still insists on having a guarantee that the DC graph obtained by RANDGEN is, in fact, good, then one can use the analysis procedures from [19] which, when given a DC graph, can determine if it is s-error detectable or not. In the extremely rare cases where the DC graph is found to be bad, one can use RANDGEN once again. Similar arguments also hold for the subsequent sections where one can check if the conditions that need to be satisfied by the DC graph are actually satisfied by it. However, this approach of verifying the "goodness" of a DC graph may, in general, be time-consuming.

IV. Error Locatability

In this section we consider the problem of error locatability. Let n be the total number of data elements, that is, $|D|$, as before. The necessary and sufficient conditions of Lemma 1.2 give us a lower bound on the number of checks required for s-error locatability [4].

Theorem 4.1: The number of checks, c, for s-error locatability is $\Omega(s \log n)$.

Proof: Clearly, from Lemma 1.2, there must be at least as many possible output patterns as there are distinct sets of errors of cardinalities at most s.

$$2^c \geq \sum_{0\leq j\leq s} \binom{n}{j} = \Omega(n^s).$$

The theorem follows. □

We suspect the preceding lower bound to be somewhat loose and think it can probably be improved.

Trivially, suppose $s = 1$. There is a simple way of achieving the lower bound of Theorem 4.1 of $\lceil \log(n+1) \rceil$ checks. We observe that the total number of distinct nonempty subsets of $\lceil \log(n+1) \rceil$ checks $(= 2^{\lceil \log(n+1) \rceil} - 1)$ is at least n. We simply connect each vertex of D, that is, each data element, to a distinct subset of the checks. One way of doing this is as follows. Let the data elements be denoted by $d_1, d_2, \cdots, d_n$ and let $q = \lceil \log(n+1) \rceil$. For a data element d_i consider the q-bit binary vector, which represents i. Then d_i would

be connected to all those checks that correspond to the 1's in the binary vector. A similar scheme was used in [12] for 2-error detection (not location). When that data element is in error, exactly the checks in the corresponding unique subset have 1's. Hence it is 1-error locatable. We should add that in addition to 1-error locatability such a DC graph also allows for easy diagnosis. The reason is that just by looking at the output pattern at the checks, we can immediately point to the data element in error. Note that the fact that this construction achieves the lower bound for $s = 1$ does not mean that this lower bound is tight. The greatest lower bound could still be $\Omega(s^2 \log n)$ (say). However, this construction does not extend in a natural way to $s \geq 2$. For example, if $s = 2$, we need to make sure that any pair of errors has distinct output patterns and this condition can be difficult to satisfy.

The main result of this section is that RANDGEN with parameters $c = (7.6s^2 + 3.8s) \log n$ and $p = 1/2s$ almost always produces an s-error-locatable DC graph. The number of checks necessary for this algorithm is quite close to the known lower bound since typically $s \ll n$. As before, we first illustrate the construction procedure by stepping through algorithm RANDGEN for our running example.

Example 4.1: Suppose that we would like to design checks such that one can locate up to three $(= s)$ errors in the data set which consists of a matrix of dimension 1024×1024. First, take 1596 checks and with each check we do the following. We consider every data element, and we include a data element in the error set of this check with a probability of one-sixth $(= 1/2s)$. At the end of this process we get a DC graph that is 3-error locating with a probability of at least $1-(1/1\,048\,576)$. It was not previously known how to design DC graphs for error location for any general value of s. However, as a basis of comparison, it must be noted that one would require 2047 checks to even locate only one error using the traditional row and column checksum method. We should note, however, that this method is deterministic, whereas ours is probabilistic. Our general method would require 760 checks to locate up to two errors and only 228 checks to locate one error. Actually, to locate only one error, we need not use this general method. From the method presented at the beginning of this section for this particular case, we need only $\lceil \log(n+1) \rceil = 21$ checks.

Before we prove our main theorem we need to state certain sufficient conditions for a DC graph to be s-error locatable.

Theorem 4.2: For every $S \subset D, |S| = 2s-1$, and for every $u \in D, u \notin S$, let there exist a check that is connected to u but not to any member of S. Then the DC graph is s-error locatable.

Proof: The conditions in the theorem are pictorially depicted in Fig. 1. Consider any two distinct subsets of D, namely R and T, such that their cardinalities are not more than s. Take any element $v \in R \oplus T$, where $\oplus$[3] represents the symmetric difference. Without loss of generality, let $v \in R$. Now by the conditions, there exists a check that is connected to v but not to any element of $R \cup T - \{v\}$, since the cardinality of this set is $\leq 2s-1$. This directly implies that this check outputs 1 when R is the set of errors and 0 when T is the set of errors; that is, these sets have different output patterns at the checks. □

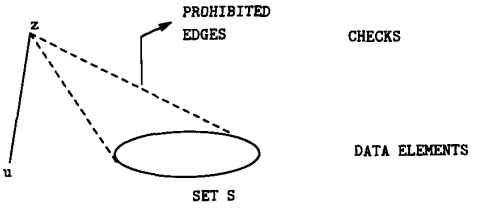


Fig. 1. Sufficient conditions for s-error locatability: For every set $S \subset D, |S| = 2s-1$ and $u \in D, u \notin S$, there exists a check z connected to u but not to any element in S.

Theorem 4.3: The algorithm RANDGEN, using $c = (7.6s^2 + 3.8s) \log n$ checks and $p = 1/2s$, produces an s-error-locatable DC graph with probability at least $1 - (1/n)$. The time complexity for constructing this graph is only $O(s^2 n \log n)$.

Proof: We will attempt to show that the DC graph satisfies the sufficient conditions of Theorem 4.2 with high probability.

Given a particular $u \in D$ and $S \subset D, u \notin S, |S| = 2s-1$, let $E_{u,S}$ be the event that no check in C satisfies the conditions of Theorem 4.2. The probability that a particular check does not satisfy the conditions is clearly $1 - p(1-p)^{2s-1}$. We now choose p so as to minimize this probability. It can be easily checked that this is minimum for $p = 1/2s$. For this value of p,

$$1 - p(1-p)^{2s-1} = 1 - \frac{1}{2s}\left(1 - \frac{1}{2s}\right)^{2s-1} \leq 1 - \frac{1}{2se}. \tag{3}$$

As the edges for each check are chosen independently, the probability that no check satisfies the conditions of Theorem 4.2, that is, $Prob(E_{u,S})$, is clearly $(1-p(1-p)^{2s-1})^c$. Observe that the probability that the DC graph does not satisfy the sufficient conditions is simply the probability that at least one of the events $E_{u,S}$ occurs, for some u and S, that is, it equals $Prob(\cup_{u,S} E_{u,S})$ where S takes all subsets of D of cardinality $2s-1$ and u takes all values in $D - S$. We bound this probability as follows:

$$\begin{aligned}
Prob(\cup_{u,S} E_{u,S}) &\leq \sum_{u,S} Prob(E_{u,S}) \\
&\leq n^{2s}(1-p(1-p)^{2s-1})^c \\
&\leq n^{2s} e^{-cp(1-p)^{2s-1}} \\
&= n^{2s} e^{-c(1/2s)[1-(1/2s)]^{2s-1}}, \quad \text{for } p = \frac{1}{2s} \\
&\leq n^{2s} e^{-\{[(4e/\log e)s^2 + (2e/\log e)s]\log n\}(1/2se)} \\
&= n^{2s}(e^{1/\log e})^{-(2s+1)\log n} \\
&= n^{2s} 2^{-\log n^{(2s+1)}}, \quad \text{since } e^{1/\log e} = 2 \\
&= n^{2s} \frac{1}{n^{2s+1}} \\
&= \frac{1}{n}
\end{aligned}$$

[3] $A \oplus B = (A - B) \cup (B - A)$.

by using (3) and choosing $c = (7.6s^2 + 3.8s)\log n \geq [(4e/\log e)s^2 + (2e/\log e)s]\log n$. Thus the probability that the DC graph is s-error locatable is at least $1 - (1/n)$. The time taken by RANDGEN follows from Theorem 2.1. □

As before, the probability that we get a bad DC graph, that is, one that is not s-error locatable, can be reduced very rapidly by adding some extra checks. By adding $3.8sk \log n$ extra checks we can decrease this probability to $1/n^{k+1}$. To illustrate through the previous example, we can add just 228 more checks to make a total of 1824 checks and our chances of producing a bad DC graph, that is, a graph that is not 3-error locating, falls very rapidly from $1/1\,048\,576$ to $1/1\,048\,576^2 \approx 1/10^{12}$. We can continue to do this, and our probability of producing a bad DC graph goes down extremely rapidly.

Corollary 4.1: The RANDGEN algorithm, using $c = (7.6s^2 + 3.8sk + 3.8s)\log n$ checks and $p = 1/2s$, produces an s-error-locatable DC graph with probability at least $1 - (1/n^{k+1})$.

A. DC Graphs with a Combination of Properties

Some researchers [20] have used DC graphs with a combination of detection and location properties for their design, that is, DC graphs that are simultaneously s-error locatable and t-error detectable. One simple way to generate these graphs is, of course, to use RANDGEN twice: once for s-error locatability as shown in this section and once for t-error detectability as shown in the previous section. By putting together the checks we would have a DC graph that is both s-error locatable and t-error detectable with a total of $(7.6s^2 + 3.8s + 3.8t)\log n$ checks. However, in many cases, this may not be necessary. Given specific values of s and t, one could choose p appropriately and calculate the minimum value of c needed to satisfy simultaneously the bounds for locatability in this section and the bounds for detectability in the previous section. This value of c may turn out to be smaller than what one would get by simply adding together the checks. However, it is difficult to give a rule of thumb in general terms. We consider next the case when $t \leq 2s$ in which we get detectability for free. This was first observed by Russel and Kime [29] in the different context of system-level diagnosis.

Lemma 4.1: Any s-error-locatable DC graph is also $2s$-error detectable.

Consider any nonempty set of data elements S of cardinality at most $2s$. We need to show that some check is at 1 if S is the set of data in error. One can always partition S into nonintersecting sets S_1 and S_2 such that the cardinalities of both the sets is less than or equal to s. Without loss of generality, let S_1 be nonempty. From the conditions of s-error locatability there must be a check, z say, that *must* be 1 when S_1 is the set of errors and *must* be 0 when S_2 is the set of errors. This means that check z is connected to exactly one element in S_1 and no element in S_2. Clearly, z must be 1 when S is the set of errors since it is connected to exactly one data element in S. Thus any set of at most $2s$ errors is detectable. □

From Lemma 4.1, we know that if $t \leq 2s$, it is sufficient to just design an s-error locatable graph using RANDGEN.

V. Designing DC-Graphs for Easy Diagnosis

We have so far concerned ourselves with the question of how to design DC graphs for s-error detectability and s-error locatability. Given that some data elements are in error, the checks take on binary values 0 or 1. Following convention, from now on we will refer to this binary vector of check outputs as the *syndrome*. An s-error-locatable DC graph assures us that no two distinct sets of errors in the data of cardinality less than or equal to s can give rise to the same syndrome. The problem of actually finding the set of data in error (or the set of processors that are faulty), given a particular syndrome, is called *diagnosis*. Note that an s-error-locatable DC graph does not necessarily imply a simple and efficient method for diagnosis. It simply assures us that given enough time and/or hardware one can eventually diagnose (locate) the errors or faults.

A straightforward, but brute-force approach, for diagnosing any syndrome in any s-error-locatable DC graph is to try all possible sets of errors of cardinality $\leq s$ and see which one is consistent with the given syndrome. Consistency of a set of errors with a syndrome is determined by checking that every check with a 0 has either no error or more than one error in its error set and every check with a 1 has one or more errors in its error set. This, of course, requires us to try $\Sigma_{1 \leq i \leq s} \binom{n}{i} = \Omega(n^s)$ different sets. For each set, checking the consistency of this set with the syndrome can be done in time proportional to the number of edges in the DC graph. This enumerative approach of trying all possible sets is too time-consuming and can be impractical even for moderate values of s. Although better results than this one are known [26], it seems quite likely that no efficient nonenumerative algorithm to diagnose an arbitrary s-error-locatable DC graph exists. This is our main motivation for designing DC graphs that are not only s-error locatable but *also allow easy diagnosis*. In the next subsection, we will propose a simple diagnosis algorithm and show how we can use RANDGEN to generate DC graphs that are not only s-error locatable but will also have the additional property that this simple diagnosis algorithm can be used to correctly locate errors in this graph.

A. Majority Diagnosis Algorithm

The majority diagnosis algorithm is a simple and intuitive algorithm, as shown below.

Majority Diagnosis Algorithm

Given a syndrome, for every data element $u \in D$ do the following.

Consider the set of all checks that are connected to data element u.

If the majority (greater than half) of these checks have a 1 then declare the data item to be erroneous.

If the majority of the checks have a 0 declare it to be error-free.

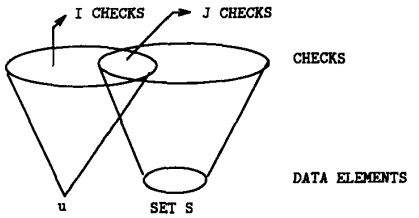


Fig. 2. Necessary and sufficient conditions for s-majority diagnosability: For every $u \in D$ and subset of the data elements $S, |S| = s, I > 2J$.

If the number of 0's equal the number of 1's, halt without giving a diagnosis.[4]

End Algorithm

This algorithm is very fast and runs in time linear to its input size, that is, in time proportional to the number of vertices and edges in the DC graph. Note that no diagnosis algorithm that has to look at the DC graph at least once before making a diagnosis can be faster than the Majority Diagnosis algorithm. In addition, each data element is decided to be erroneous or not independently of the others and, therefore, the Majority Diagnosis algorithm can also be executed extremely efficiently in parallel. However, this algorithm will not, of course, diagnose correctly for any general s-error-locating DC graph. The challenge would be to *synthesize s-error-locating DC graphs with the added property that this majority algorithm can be used to diagnose correctly the errors*. In addition, could we do it without sacrificing the results of Section IV, which are nearly asymptotically optimal? We answer this question in the affirmative in Theorem 5.1 by showing that only a constant factor overhead is needed to accomplish this. We first define some concepts.

Definition 5.1: A DC graph is said to be s-majority diagnosable iff the Majority Diagnosis algorithm can correctly locate any set of s or fewer errors from the syndrome.

Lemma 5.1: A DC graph is s-majority diagnosable iff for ever $u \in D$ and $S \subset D, |S| = s$, the following is true: The cardinality of the set of checks connected to u but not to any data element in $S - \{u\}$ is greater than the cardinality of the set of checks which are simultaneously connected to u and some nonempty subset of $S - \{u\}$.

Proof: Let I be the number of checks connected to the data element u and J be the number of checks simultaneously connected to u and some nonempty subset of $S - \{u\}$. The pictorial representation of the conditions is shown in Fig. 2 when $u \notin S$. Note that the conditions imply that $I - J > J$, that is, $I > 2J$. We first prove the necessity of the conditions.

Only if: Assume that we have an s-majority-diagnosable graph. For contradiction, assume that there is a set $S \subset D$ of cardinality s and a data element u such that the condition is not satisfied. In the proof below, we will not consider the case when some data element has the same number of checks with 0 as the number of checks with 1, since in this case the diagnosis algorithm will fail anyway.

[4]This step is just a technical detail to keep the proofs simple. Our DC graph construction method given later avoids this situation.

There are two possibilities. If $u \in S$, then consider the error pattern in which all data elements of S are in error and the remaining elements are error-free. Every check connected to only u and no other element of S necessarily outputs 1. The other checks connected to u *could* all output 0 since they are connected to at least two erroneous data elements. If the condition is not satisfied, the number of 0 checks connected to u exceeds the 1 checks connected to u. Thus the majority algorithm will falsely diagnose data element u as error-free for this syndrome. This is a contradiction.

Now suppose $u \notin S$. Again consider the situation when all the data elements in S are erroneous and the remaining elements error-free. The checks connected only to u and not to any element in S are necessarily 0. But the checks simultaneously connected to u and to some nonempty subset of S *could* all be 1 since they are connected to at least one data element with error. If the condition is not satisfied, the number of checks with output 1 connected to u exceeds the number of checks with output 0 connected to u. Thus the majority algorithm will wrongly diagnose u to be erroneous with this syndrome. This is a contradiction.

If: Now we prove that if the conditions are met then the DC graph is s-majority diagnosable. Suppose we are given a set of errors T. Without loss of generality, we can assume $|T| = s$. From the conditions, we know that the number of checks connected to a data element u and not to any element of $T - \{u\}$ is greater than the number of checks simultaneously connected to u and some nonempty subset of $T - \{u\}$. The former set of checks always takes 1 or 0, depending on whether u is in the set of errors T. Thus the majority of the checks connected to u necessarily have values 1 or 0, respectively.□

Example 5.1: To illustrate the concepts we give an example of a 1-majority diagnosable DC graph. Consider a set of 64 data elements arranged in a $4 \times 4 \times 4$ cube. We associate a check with the data elements in the same row either in the x, y, or z coordinate axis. So there are a total of 48 checks, and each data element is connected to three checks in each of the three coordinate axes. Given any data element u and any other data element v, the number of checks connected to both u and v is at most 1, that is, when u and v are in the same row along either the x, y or the z axis. Since this is always less than the number of checks connected just to u and not to v, this arrangement is 1-majority diagnosable.

We now show the distinction between majority diagnosability and just error locatability. It can be seen that the DC graph in this example is 2-error locatable. A quick way to see this is to note that the following funny algorithm always diagnoses up to two errors uniquely and correctly. For each data element, compute the number of checks connected to it that output a 1. Declare the data elements with the maximum such nonzero number as erroneous and the rest as error-free! However, the above example is not 2-majority diagnosable. To see this, assume that the data elements in positions $(x, y, z + 1)$ and $(x, y + 1, z)$ are in error and the others are error-free. Data element (x, y, z) will now be connected to two checks at 1 and only one check at 0. The majority diagnosis algorithm will incorrectly declare (x, y, z) to be in error!

Note that an s-majority-diagnosable DC graph is automatically s-error locatable since by Definition 5.1 the Majority Diagnosis algorithm correctly and uniquely diagnoses any set of errors of cardinality s or less from the syndrome. Another way to see this is to observe that the conditions in Lemma 5.1 imply the sufficient conditions for s-error locatability in Theorem 4.2.

Before we prove the main theorem of this section, we need a few lemmas from probability theory. Bounds of this nature were first reported in [6]. The following *Generalized Chernoff bounds* are from [22]. Intuitively, these lemmas state that a sum of independent Boolean random variables is very unlikely to take values "far" from its mean. A special case of this result we can all readily associate with is that the number of heads in m independent coin tosses of a far coin tends to be close to $m/2$.

Lemma 5.2: Let $X_1, X_2, \cdots, X_m$ be independent Boolean random variables with $Prob(X_i = 1) = p$ and $Prob(X_i = 0) = 1 - p$. Let $X = \Sigma_{1 \le i \le m} X_i$ and $\delta > 0$. Then

$$Prob(X > (1+\delta)\mu(X)) \le \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu(X)}$$

where $\mu(X)$, the expected (or average) value of X, equals mp.

Lemma 5.3: Let $X_1, X_2, \cdots, X_m$ be independent Boolean random variables with $Prob(X_i = 1) = p$ and $Prob(X_i = 0) = 1 - p$. Let $X = \Sigma_{1 \le i \le m} X_i$ and $\delta > 0$. Then

$$Prob(X \le (1-\delta)\mu(X)) \le \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu(X)}$$

where $\mu(X)$, the expected (or average) value of X, equals mp.

Theorem 5.1: The algorithm RANDGEN, using $c = 60.5(s^2 + 2s) \log n$ checks, $s > 1$, and $p = 1/8s$, produces a DC graph that is s-majority diagnosable with probability at least $1 - (1/n)$.

Proof: We need to show that all the conditions of Lemma 5.1 are met with a high probability. As before, we try to show that the probability that one of these conditions is not met is extremely small. Let the event $E_{u,S}$ represent the event that the conditions are *not* met for $u \in D$ and $S \subset D, |S| = s$, that is, that the cardinality of the set of checks connected to u is less than or equal to twice the cardinality of the set of checks connected simultaneously to u and some nonempty subset of $S - \{u\}$. The probability that we get a bad DC graph, that is, one that is not s-majority diagnosable, simply equals $Prob(\cup_{u,S} E_{u,S})$, where u takes on all values in D and S takes on all subsets of D of cardinality s.

Given a data element $u \in D$ and $S \subset D, |S| = s$, we bound $Prob(E_{u,S})$ as follows. We will assume that $u \notin S$. The case when $u \in S$ is similar. Let I be the cardinality of the set of checks connected to data element u. Let J be the cardinality of the set of checks connected to data element u as well as some nonempty subset of $S - \{u\}$ ($= S$ for this case). Note that both I and J can be expressed as the sum of independent binary random variables. $I = \Sigma_i X_i$, where X_i is 1 if check i is connected to u and 0 otherwise,

$$\mu(I) = \mu\left(\sum_i X_i\right) = \sum_{1 \le i \le c} \mu(X_i) = pc.$$

Similarly, $J = \Sigma_i Y_i$, where Y_i is 1 if check i is simultaneously connected to u and some nonempty subset of S and 0 otherwise. It can be seen that $\mu(Y_i) = p(1 - (1-p)^s)$. Thus

$$\mu(J) = \mu\left(\sum_i Y_i\right) = p(1 - (1-p)^s)c.$$

Now clearly[5]

$$\begin{aligned} Prob(E_{u,S} &= Prob(2J \ge I) \\ &\le Prob(I \le (1-\alpha)\mu(I)) \\ &\quad + Prob(2J \ge I | I > (1-\alpha)\mu(I)) \\ &\le Prob(I \le (1-\alpha)\mu(I)) \\ &\quad + Prob(2J > (1-\alpha)\mu(I)) \end{aligned} \tag{4}$$

where α is chosen to be 0.3968.[6] Now we use Lemma 5.3 to bound the first term in (4),

$$\begin{aligned} &Prob(I \le (1-\alpha)\mu(I)) \\ &\quad \le \left(\frac{e^{-\alpha}}{(1-\alpha)^{1-\alpha}}\right)^{\mu(I)} \\ &\quad = e^{-\{\alpha + (1-\alpha)[\log(1-\alpha)/\log e)]\}pc} \\ &\quad \le e^{-0.09187pc} \\ &\quad \le e^{-0.01148(c/s)}, \quad \text{for } p = \frac{1}{8s} \\ &\quad \le \frac{1}{n^{s+2}} \end{aligned} \tag{5}$$

using $c = 60.5(s^2 + 2s) \log n$. Observe that for $s > 1$,

$$\begin{aligned} \frac{\mu(I)}{\mu(2J)} &= \frac{1}{2\left(1 - \left(1 - \frac{1}{8s}\right)^s\right)} \\ &\ge \frac{1}{2\left(1 - \left(1 - \frac{1}{16}\right)^2\right)} \\ &\ge 4.129. \end{aligned} \tag{6}$$

Now we use Lemma 5.2 to bound the second term in (4), where

$$\begin{aligned} &Prob(2J > (1-\alpha)\mu(I)) \\ &\quad = Prob\left(2J > \frac{(1-\alpha)\mu(I)}{\mu(2J)} 2\mu(J)\right) \\ &\quad \le Prob(J > (1+1.49)\mu(J)) \\ &\quad \le \left(\frac{e^{1.49}}{(1+1.49)^{1+1.49}}\right)^{\mu(J)} \\ &\quad \le e^{-0.7815(0.1175pc)} \\ &\quad \le e^{-0.01147(c/s)}, \quad \text{for } p = \frac{1}{8s} \\ &\quad \le \frac{1}{n^{s+2}} \end{aligned} \tag{7}$$

using (6), the fact that $\mu(J) \ge (1 - e^{-(1/8)})pc$ and finally substituting $c = 60.5(s^2 + 2s) \log n$.

[5] In what follows, $Prob(E|F)$ denotes the conditional probability of event E given event F [11].

[6] This number was chosen to optimize the bounds that follow.

The proof of these bounds for $Prob(E_{u,S})$ when $u \in S$ is similar. Each event $E_{u,S}$ gives rise to two terms and there are $n\binom{n}{s}$ events in all. Thus the probability of a bad DC graph is

$$Prob(\cup E_{u,S}) \leq 2n\binom{n}{s}\frac{1}{n^{s+2}} \leq 2n\frac{n^s}{s!}\frac{1}{n^{s+2}}$$
$$\leq \frac{1}{n}, \quad \text{for } s > 1. \qquad \square$$

In the preceding theorem, we considered s-majority-diagnosable DC graphs for $s > 1$. A similar result can be obtained easily for the case when $s = 1$. However, it should be pointed out that we have already given a trivial and efficient method at the beginning of Section IV for obtaining a 1-error locatable DC graph which also allows easy diagnosis to be done.

We can reduce the probability even further, as before, of getting a bad DC graph to $1/n^k$ by using $c = 60.5(s^2 + ks + s) \log n$ checks.

VI. Uniform Checks

It has been noted in [28] that if all the checks have the same error-detection capability and check the same number of data elements then their design is simplified. Another advantage of such uniform checks is that their hardware and time overheads can also be uniform. Note that in DC graphs produced by RANDGEN, two checks could possibly have different error set cardinalities and hence not be uniformly identical. One can easily modify RANDGEN to make it produce uniform checks. In RANDGEN we randomly included a data element in the error set of a check with some probability p. Instead, for every check we now simply pick as its error set a random subset of the data of a fixed cardinality leading to an algorithm called UNIFGEN (UNIForm GENeration). This algorithm has two parameters: c, the number of checks, and g, the cardinality of the error set of the uniform checks. Similar to RANDGEN, by simply varying its two parameters, UNIFGEN can generate DC graphs with uniform checks having a variety of useful properties. As we shall soon see, this fixed cardinality g must be of the order of the mean cardinality of the error set of the checks in the corresponding DC graphs produced by RANDGEN.

UNIFGEN(c, g)
Let D be the set of data elements and C be the set of checks such that $|C| = c$.
For every check, pick uniformly and at random a subset of the data of cardinality g. This will be the error set of this check. Do this for every check.
End UNIFGEN

We first consider the problem of generating uniform checks for s-error detection. We will see that requiring "uniform" checks costs us nothing; that is, UNIFGEN uses the same number of checks as RANDGEN for generating s-error-detectable DC graphs.

Theorem 6.1: The algorithm UNIFGEN, using $c = 3.8s \log n$ checks and $g = n/s$, produces an s-error-detectable DC graph with probability at least $1 - [1/(n-1)]$.

Proof: The proof is similar to that of Theorem 3.2, and notations from the proof of that theorem will be used here. As before we need to show that the DC graph satisfies the conditions of Lemma 1.1, that is, every set $S \subset D$, $|S| \leq s$, has a check z such that it is connected to exactly one element of S. For any single set S, $|S| = i \leq s$, and a particular check z, we first evaluate the probability that z is not connected to exactly one element of S. The total number of ways of choosing the error set of check z is clearly $\binom{n}{\frac{n}{s}}$. Of these the number of subsets that contain exactly one element from S is clearly $\binom{n-i}{\frac{n}{s}-1}$. Hence the probability that the check is not connected to exactly one element of S is[7]

$$1 - i\frac{\binom{n-i}{(n/s)-1}}{\binom{n}{n/s}}$$
$$= 1 - \frac{i}{s}\frac{(n-i)^{\underline{(n/s)-1}}}{(n-1)^{\underline{(n/s)-1}}}$$
$$\leq 1 - \frac{i}{s}\left(1 - \frac{i-1}{n-(n/s)+1}\right)^{(n/s)-1}$$
$$\leq 1 - \frac{i}{s}\left(1 - \frac{s}{n}\right)^{(n/s)-1}$$
$$\leq 1 - \frac{i}{se}. \qquad (8)$$

This is the same as what we had earlier in (1). The rest of the proof is similar to that of Theorem 3.2. $\square$

We can, of course, reduce the probability of getting a bad DC graph, that is, one that is not s-error detectable, rapidly to less than or equal to $1/(n^{k+1} - 1)$ by using $(3.8s + 1.9sk) \log n$ checks, as before.

We now turn to the problem of generating an s-error-locatable DC graph with uniform checks. Unlike the case of error detection, UNIFGEN does have an overhead of a small constant factor in terms of the required number of checks when compared with RANDGEN for this problem.

Theorem 6.2: The algorithm UNIFGEN, using $c = (10.6s^2 + 5.3s) \log n$ checks and $g = n/2s$, produces an s-error-locatable DC graph with probability at least $1 - (1/n)$, when $n \geq 2.8s$.

Proof: The proof is similar to that of Theorem 4.3 and notations from the proof of that theorem will be used here. We need to show that the DC graph satisfies the conditions of Theorem 4.2. We evaluate the probability that a particular check does not satisfy the conditions as follows. The total number of ways of choosing a subset of size $n/2s$ is clearly $\binom{n}{\frac{n}{2s}}$. The number of subsets that contain an element u but no element from set S, $|S| = 2s - 1$, is clearly $\binom{n-2s}{\frac{n}{2s}-1}$. Thus the probability that a particular check does not satisfy

[7] $x^{\underline{i}}$ denotes the ith falling power of x, that is, $x(x-1)\cdots(x-i+1)$.

the conditions of Theorem 4.2 is

$$\begin{aligned}
&1 - \frac{\binom{n-2s}{(n/2s)-1}}{\binom{n}{n/2s}} \\
&\quad = 1 - \frac{1}{2s}\frac{(n-2s)^{\underline{(n/2s)-1}}}{(n-1)^{\underline{(n/2s)-1}}} \\
&\quad \le 1 - \frac{1}{2s}\left(1 - \frac{2s-1}{n-(n/2s)+1}\right)^{(n/2s)-1} \\
&\quad \le 1 - \frac{1}{2s}\left(1 - \frac{8s}{3n}\right)^{(n/2s)-1} \\
&\quad \le 1 - \frac{1}{2se^{4/3}} \qquad (9)
\end{aligned}$$

when $n \ge 2.8s$. Compare this with what we had earlier in (3). Now we continue in a manner similar to the proof of Theorem 4.3. We need to select $c = (10.6s^2 + 5.3s)\log n \ge [4e^{4/3}/\log e)s^2 + (2e^{4/3}/\log e)s]\log n$ to complete the proof. □

As before, we can drastically reduce the probability of getting a bad DC graph, that is, one that is not s-error locatable, to less than or equal to $1/n^{k+1}$ by using $(10.6s^2 + 5.3sk + 5.3s)\log n$ checks.

VII. Check Computation Failures

In this section, we consider the more realistic case when the dedicated processors computing the checks can themselves fail. Note that only the processors doing data computations are represented in P and the check-computing processors are not represented in the PDC graph. In this section, we will distinguish between the (actual) output value of a check and the *correct output value* of a check. The correct output value of a check is simply the output of the check, had the check computation been error-free. The output of a check could be different from its correct output due to faults in the processors that compute it. One now needs to modify the standard definition of a t-fault-detectable/locatable PDC graph to incorporate this possibility.

Definition 7.1: A PDC graph is said to be strictly t-fault detectable/locatable iff any set of at most t faults in the processors in P can be detected/located in spite of at most t faults in the check-computing processors.

How can we modify the various procedures for synthesizing PDC graphs mentioned in Section I under these stricter assumptions? It is customary to add more processors when one tries to make a nonfault tolerant computation fault tolerant. For example, if our chosen architecture is a 2-D mesh of processors, a natural approach would be to implement the fault tolerant computation on a mesh that has one more row and column than that required by the non-fault tolerant version. That is, we would add $2\sqrt{|P|} + 1$ extra processors, where $|P|$ is the number of processors in the original non-fault tolerant mesh computation. Note that the number of checks necessary in our DC graph designs in Sections III–VI is small, that is, it grows only as a logarithm of the number of data elements. Therefore, in the case that we have a sufficiently large number of extra processors, one could assign different sets of $2t+1$ processors to compute independently the different checks and take the majority value as the output of any given check. Clearly this $(2t+1)$-modular redundancy in the check computations makes sure that check computations are always correct as long as not more than t of the check-computing processors fail. Since now the check computations are always correct, we could use the old synthesis techniques outlined in Section I with the result that the synthesized system is strictly t-fault detectable/locatable as specified in Definition 7.1.

In the event that we do not have as many as $2t+1$ extra processors for each check computation, we would have to assume that each check is computed by a single unique dedicated processor. Therefore, in our definitions for s-error detectability or locatability of DC graphs we would have to consider the possibility of the checks themselves being erroneous. In this stricter sense, one could define a DC graph to be *strictly s-error detectable/locatable* iff it can detect/locate any set of at most s errors in the data even if any set of at most s check computations are erroneous. It is clear that now if one is to use strictly s-error detectable/locatable graphs in the PDC graph design procedures in place of simple s-error detectable/locatable DC graphs, we would obtain PDC graphs with the same amount of fault tolerance but in the stricter sense of Definition 7.1.

In the rest of this section, we show how our results can be extended easily by using RANDGEN to produce strictly s-error-detectable, strictly s-error-locatable, as well as strictly s-majority-diagnosable DC graphs.

A. Strict Error Detectability

In this subsection, we show how to construct strictly s-error-detectable graphs.

Lemma 7.1: A DC graph is strictly s-error detectable iff every possible nonempty set of errors in the data of cardinality at most s makes at least $2s+1$ of the checks have their correct outputs as 1.

Proof: If there is no error in the data then there can be at most s checks that output 1, since at most s checks could be erroneous. Therefore, if there are some, not more than s, errors in the data, then at least $2s+1$ of the checks must have their correct outputs as 1. Of these at most s checks can output a 0 due to erroneous check computations. Therefore, there are at least $s+1$ checks that output a 1. Thus if there are $\le s$ checks outputting a 1, then we can conclude that there is no error in the data. Else there is some error in the data. □

We now show that RANDGEN can also produce strictly s-error-detectable DC graphs.

Theorem 7.1: The algorithm RANDGEN, using parameter $c = 11.31s\log n$ and $p = 1/s$, produces a strictly s-error-detectable DC graph with probability at least $1 - [1/(n-1)]$, for sufficiently large n.

Proof: The proof is similar to that of Theorem 3.2. We need to show that the DC graph satisfies the conditions of Lemma 7.1. As derived in (1), for any single set $S, |S| = i$, and a particular check z, the probability that z is not connected to exactly one element of S is $\le 1 - i(1/se)$. Let E_S be the

event that less than $2s+1$ checks are connected to exactly one element in S, where S is any nonempty set of cardinality i.

$$\begin{aligned} Prob(E_S) &\le \sum_{1\le j\le 2s} \binom{c}{c-j}\left(1-i\frac{1}{se}\right)^{c-j} \\ &= \sum_{1\le j\le 2s} \binom{c}{j}\left(1-i\frac{1}{se}\right)^{c-j} \\ &\le \sum_{1\le j\le 2s} c^j e^{-i(1/se)(c-j)} \\ &\le e^{-(ic/se)} \sum_{1\le j\le 2s} c^j e^{(ij/se)} \\ &\le e^{-(ic/se)} \sum_{1\le j\le 2s} c^j e^j \\ &\le 2c^{2s}e^{2s}e^{-(ic/se)} \\ &\le \frac{1}{n^{2i}} \end{aligned} \tag{10}$$

using the fact that $\binom{c}{c-j} = \binom{c}{j} \le c^j$ bounding a geometric series by twice the last term since $ce \ge 2$, choosing $c = 11.32s \log n$ and assuming that $\log n \ge 3s + s[(\log s + \log\log n)/2]$. The calculations for the last condition are tedious but straightforward and are given in the appendix. The rest of the proof is the same as in Theorem 3.2. □

It should also be noted that the condition in the preceding proof that requires that n to be greater than some value can be relaxed easily. The specific value of c and the corresponding condition on n in the theorem are only meant to be typical illustrations. One could always make the algorithm work for smaller ranges of n by sufficiently increasing the constant factor involved in c. Let $c = \hat{c}s \log n$ and let the smallest value in our range of interest for n be n_0. It is sufficient that constant $\hat{c}$ be large enough such that the last inequality of (10) holds true for every value of n in the range of interest. This implies the following sufficient condition for the constant $\hat{c}$. The calculations required to derive this equation are shown in the appendix.

$$\begin{aligned} &\frac{\hat{c}\log e}{e} - \frac{2s\log\hat{c}}{\log n_0} \\ &\quad \ge 2 + \frac{1 + 2s\log s + 2s\log\log n_0 + 2s\log e}{\log n_0}. \end{aligned} \tag{11}$$

It is possible to generate strictly s-error-detectable DC graphs with uniform checks using UNIFGEN. A theorem similar to Theorem 7.1 for UNIFGEN is stated next.

Theorem 7.2: The algorithm UNIFGEN, using parameter $c = 11.31s\log n$ and $g = n/s$, produces a strictly s-error-detectable DC graph with probability at least $1 - [1/(n-1)]$, for sufficiently large n.

Proof: As derived in (8), for any single set $S, |S| = i$, and a particular check z, the probability that z is not connected to exactly one element of S is $\le 1 - i(1/se)$. The rest of the proof is the same as that of Theorem 7.1. □

B. Strict Error Locatability

We now show how to construct strictly s-error-locatable DC graphs.

Lemma 7.2: A DC graph is strictly s-error locatable iff the following is true. Consider any pair of sets of errors of cardinality at most s. There must be at least $2s + 1$ checks whose correct output values are necessarily different for these two sets of errors.

Proof: The conditions imply that no two sets of errors of cardinality at most s can produce the same output pattern at the checks. This is so because at most s of the check values for either set of errors could be erroneous. Therefore, at most $2s$ check outputs could be made to agree with one another due to check computation errors. □

As before, we next state sufficient conditions for strict s-error locatability, which will form the basis of a later theorem.

Theorem 7.3: For every $S \subset D, |S| = 2s-1$, and for every $u \in D, u \notin S$, let there exist $2s + 1$ checks, each of which is connected to u but not to any member of S. Then the DC graph is strictly s-error locatable.

Proof: The proof is similar to Theorem 4.2. Consider any two distinct subsets of D, namely R and T, such that their cardinalities are not more than s. Take any element $v \in R \oplus T$. Without loss of generality, let $v \in R$. Now by the conditions, there exist $2s+1$ checks, each of which is connected to v but not to any element of $R \cup T - \{v\}$.This directly implies that these checks have their correct outputs to be 1 when R is the set of errors and 0 when T is the set of errors, that is, there are $2s+1$ checks whose correct output values differ for R and T. From Lemma 7.2, this implies that the DC graph is strictly s-error locatable. □

We now show that RANDGEN can produce strictly s-error locatable DC graphs.

Theorem 7.4: The algorithm RANDGEN, using $c = (15.2s^2 + 3.8s)\log n$ checks and $p = 1/2s$, produces a strictly s-error-locatable DC graph with probability at least $1-(1/n)$, for sufficiently large n.

Proof: We will attempt to show that the DC graph satisfies the sufficient conditions of Theorem 7.3 with high probability. The proof is very similar to the proof of Theorem 4.3. Given a particular $u \in D$ and $S \subset D, u \notin S, |S| = 2s-1$, let $E_{u,S}$ be the event that no set of $2s + 1$ checks satisfies the conditions of Theorem 7.3. As derived earlier in (3), the probability that a particular check does not satisfy the conditions is at most $1 - (1/2se)$. Note that if event $E_{u,S}$ is true, then there must exist some set of $c - 2s$ checks which does not satisfy the sufficient conditions of Theorem 7.3. This gives us our first inequality below.

$$\begin{aligned} Prob(E_{u,S}) &\le \binom{c}{c-2s}\left(1-\frac{1}{2se}\right)^{c-2s} \\ &= \binom{c}{2s}\left(1-\frac{1}{2se}\right)^{c-2s} \\ &\le c^{2s}e^{-(1/2se)(c-2s)} \\ &= c^{2s}e^{1/e}e^{-(c/2se)} \\ &\le \frac{1}{n^{2s+1}} \end{aligned} \tag{12}$$

choosing $c = (15.2s^2 + 3.8s)\log n$ and assuming that $\log n \geq 4.6 + 2\log s + \log\log n$. The calculations for the last condition are similar to the calculations presented in the appendix for Theorem 7.1. The rest of the proof is the same as in Theorem 4.3. □

As before, we can use the algorithm for smaller ranges of n by increasing the constant involved in the value of c such that the last inequality of (12) is satisfied. Let n_0 be the smallest value of n in our range of interest and let $c = (\hat{c}s^2 + 3.8s)\log n$. For our construction to work for all values of n in our range of interest, it is sufficient to choose constant $\hat{c}$ such that the following inequality is satisfied. The calculations are similar in spirit to those presented in the appendix for the previous subsection.

$$\frac{\hat{c}}{3.8} - \frac{2\log(\hat{c}+3.8)}{\log n_0} \geq 2 + \frac{4\log s + 2\log\log n_0 + 0.54/s}{\log n_0}. \quad (13)$$

It is possible to generate strictly s-error-locatable DC graphs with uniform checks using UNIFGEN. A theorem similar to Theorem 7.4 for UNIFGEN is stated next.

Theorem 7.5: The algorithm UNIFGEN, using $c = (21.3s^2 + 5.31s)\log n$ checks and $p = 1/2s$, produces a strictly s-error-locatable DC graph with probability at least $1-(1/n)$, for sufficiently large n.

Proof: Given a particular $u \in D$ and $S \subset D, u \notin S, |S| = 2s-1$, let $E_{u,S}$ be the event that no set of $2s+1$ checks satisfies the conditions of Theorem 7.3. As derived earlier in (9), the probability that a particular check does not satisfy the conditions is at most $1-(1/2se^{4/3})$. The rest of the proof is similar to the proof of Theorem 7.4. The condition on n for the value of c chosen in the theorem is slightly different. It is $\log n \geq 4.92 + 2\log s + \log\log n$. □

C. Strict Majority Diagnosability

We can also extend the notion of majority diagnosability to the case when check computations themselves can become erroneous. Intuitively, a DC graph is *strictly s-majority diagnosable* if one can apply the majority diagnosis algorithm correctly to diagnose s or fewer data errors even when s or fewer check computations are erroneous.

Lemma 7.3: A DC graph is strictly s-majority diagnosable iff for every $u \in D$ and $S \subset D, |S| = s$, the following is true: The cardinality of the set of checks connected to u but not to any data element in $S - \{u\}$ exceeds the cardinality of the set of checks that are simultaneously connected to u and some nonempty subset of $S - \{u\}$ by at least $2s+1$.

Proof: The proof is similar to that of Lemma 5.1. The margin of $2s+1$ makes sure that the correct majority decision cannot be swayed by erroneous check computations. □

Theorem 7.6: The algorithm RANDGEN, using $c = 95(s^2+2s)\log n$ checks, $s > 1$, and $p = 1/8s$, produces a DC graph that is strictly s-majority diagnosable with probability at least $1-(1/n)$, for large enough values of n.

Proof: The proof is similar to that of Theorem 5.1. As before, let the event $E_{u,S}$ represent the event that the conditions of Lemma 7.3 are *not* met for $u \in D$ and $S \subset D$, $|S| = s$. Given a data element $u \in D$ and an $S \subset D$, $|S| = s$, we bound $Prob(E_{u,S})$ as follows. We will assume that $u \notin S$. The case when $u \in S$ is similar. Sets I and J are defined as earlier.

$$\begin{aligned} Prob(E_{u,S}) &= Prob(2J + 2s + 1 > I) \\ &\leq Prob(I \leq (1-\alpha)\mu(I)) \\ &\quad + Prob(2J + 2s \geq I | I > (1-\alpha)\mu(I)) \\ &\leq Prob(I \leq (1-\alpha)\mu(I)) \\ &\quad + Prob(2J + 2s > (1-\alpha)\mu(I)) \\ &\leq Prob(I \leq (1-\alpha)\mu(I)) \\ &\quad + Prob(2J > (1-\alpha)\mu(I) - 2s) \end{aligned} \quad (14)$$

where α is chosen as before to be 0.3968. The first term in (14) can be bound as before to be at most $1/n^{s+2}$. Noting that $\mu(I) = pc = c/8s$ and for $s > 1$

$$\begin{aligned} &\frac{\mu(I) - (1-\alpha)^{-1}2s}{\mu(2J)} \\ &\geq \frac{1}{2\left(1-\left(1-\frac{1}{8s}\right)^s\right)} - \frac{26.53s^2}{2\left(1-\left(1-\frac{1}{8s}\right)^s\right)c} \\ &\geq \frac{1}{2\left(1-\left(1-\frac{1}{16}\right)^2\right)} - \frac{1.1532}{\log n} \\ &\geq 4.0137 \end{aligned} \quad (15)$$

assuming that $\log n \geq 10$.

As previously, we use Lemma 5.2 to bound the second term in (14).

$$\begin{aligned} &Prob(2J > (1-\alpha)\mu(I) - 2s) \\ &= Prob\left(2J > (1-\alpha) \cdot \frac{\mu(I) - (1-\alpha)^{-1}2s}{\mu(2J)} 2\mu(J)\right) \\ &\leq Prob(J > (1+1.421)\mu(J)) \\ &\leq \left(\frac{e^{1.421}}{(1+1.421)^{1+1.421}}\right)^{\mu(J)} \\ &\leq e^{-0.7196(0.1175pc)} \\ &\leq e^{-0.010569(c/s)}, \quad \text{for } p = \frac{1}{8s} \\ &\leq \frac{1}{n^{s+2}}. \end{aligned} \quad (16)$$

From this it follows that the probability of getting a bad DC graph, that is, one that is not s-majority diagnosable, is

$$Prob(\cup E_{u,S}) \leq 2n\binom{n}{s}\frac{1}{n^{s+2}} \leq 2n\frac{n^s}{s!}\frac{1}{n^{s+2}} \leq \frac{1}{n}, \quad \text{for } s > 1.$$

□

VIII. Conclusions

In this paper, we proposed a simple and efficient general-purpose algorithm for generating arbitrary data-check (DC) graphs with a small number of checks, which satisfy a variety of properties that have been found to be useful in algorithm-based fault tolerance (ABFT) designs. Although we have stated the results in the context of ABFT, we feel that the techniques and ideas used here will also be useful in the context of other fault tolerance problems. We introduced the concept of majority diagnosability in an attempt toward explicitly designing DC graphs for easy diagnosis. We believe this to be a good example of how one can simplify many issues by simply restricting the space of possible designs. Of course, we need to be sure that this does not increase the overhead of the design too drastically and also that we have a good procedure to construct such designs. We showed that RANDGEN served our purpose on these counts. We also examined UNIFGEN, a variation of RANDGEN, that produced DC graphs with uniform checks.

Finally, our constructions were probabilistic and necessarily have a small probability of not producing a DC graph with the required properties. As noted earlier, one can decrease this probability very rapidly by adding some extra checks. In fact, in general, fixing the probability of getting a "bad" DC graph that one is willing to tolerate, one can calculate what value of c we need to use. It should be pointed out that even in the extremely rare cases where one gets a bad DC graph, the construct will still detect/locate most sets of s or fewer errors.

Appendix A
Calculation for Conditions in Theorem 7.1

We need to show that the last inequality in (10) is true if $\log n \geq 3s + s[(\log s + \log\log n)/2]$. The last inequality is

$$2c^{2s}e^{2s}e^{-(ic/se)} \leq 1/n^{2i}.$$

Taking logarithms on both sides and substituting for c, we have

$$1 + 2s\log 11.31 + 2s\log s + 2s\log\log n + 2s\log e \leq i\left(\frac{11.31s\log n\log e}{se} - 2\log n\right).$$

Now since $1 \leq i \leq s$ the preceding condition will be true for all i if it is true for $i = 1$. Substituting $i = 1$ we have

$$1 + 2s\log 11.31 + 2s\log s + 2s\log\log n + 2s\log e \leq (6\log n - 2\log n).$$

Now simplifying further, the preceding inequality is satisfied if

$$4\log n \geq 1 + 9.885s + 2s\log s + 2s\log\log n.$$

Now $1 + 9.885s \leq 12s$, since $s \geq 1$. Using this we know that the preceding condition is satisfied if

$$\log n \geq 3s + s\frac{(\log s + \log\log n)}{2}.$$

Appendix B
Calculation for Deriving the Condition on Constant $\hat{c}$ in Section VII-A

For the procedure to work for all $n \geq n_0$ we need to show that the last inequality in (10) is true if

$$\frac{\hat{c}\log e}{e} - \frac{2s\log\hat{c}}{\log n_0} \geq 2 + \frac{1 + 2s\log s + 2s\log\log n_0 + 2s\log e}{\log n_0}$$

where $c = \hat{c}s\log n$. As before, taking logarithms on both sides of the last inequality in (10), substituting for c and setting $i = 1$ and $n = n_0$ we get the following sufficient condition:

$$1 + 2s\log\hat{c} + 2s\log s + 2s\log\log n_0 + 2s\log e \leq \left(\frac{\hat{c}\log e}{e}\log n_0 - 2\log n_0\right).$$

Dividing both sides by $\log n_0$ and rearranging the terms such that all terms having $\hat{c}$ appear in the left-hand side, we have the required condition $\hat{c}$.

References

[1] J. A. Abraham *et al.*, "Fault tolerance techniques for systolic arrays," *IEEE Computer,* pp. 65–74, July 1987.

[2] B. Bollobas, *Random Graphs.* New York: Academic Press, 1985.

[3] P. Banerjee *et al.*, "An evaluation of system-level fault tolerance on the Intel hypercube multiprocessor," in *Proc. Int. Symp. Fault Tolerant Comput.*, Tokyo, June 1988, pp. 362–367.

[4] P. Banerjee and J. A. Abraham, "Bounds on algorithm-based fault tolerance in multiple processor systems," *IEEE Trans. Comput.*, vol. C-35, pp. 296–306, Apr. 1986.

[5] P. Banerjee and J. A. Abraham, "A probabilistic model of algorithm-based fault tolerance in array processors for real-time systems," in *Proc. Real-Time Systems Symp.*, 1986, pp. 72–78.

[6] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Math. Stat.*, vol. 23, pp. 493–509, 1952.

[7] C-Y. Chen and J. A. Abraham, "Fault-tolerant systems for the computation of eigenvalues and singular values," in *Proc. SPIE Adv. Alg. Arch. Signal Proc.*, vol. 696, pp. 228–237, Aug. 1986.

[8] Y-H. Choi and M. Malek, "A fault tolerant FFT processor," *IEEE Trans. Comput.*, vol. C-37, pp. 617–621, May 1988.

[9] Y-H. Choi and M. Malek, "A fault tolerant systolic sorter," *IEEE Trans. Comput.*, vol. C-37, pp. 621–624, May 1988.

[10] P. Erdos and J. Spencer, *The Probabilistic Method in Combinatorics.* New York: Academic Press, 1974.

[11] W. Feller, *An Introduction to Probability Theory and its Applications,* vol. I. New York: John Wiley, 1968.

[12] D. Gu, D. J. Rosenkrantz, and S. S. Ravi, "Design and analysis of test schemes for algorithm-based fault tolerance," in *Proc. Int. Symp. Fault Tolerant Comput.*, Newcastle-upon-Tyne, U.K., June 1990, pp. 106–113.

[13] K-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, pp. 518–528, June 1984.

[14] J-Y. Jou and J. A. Abraham, "Fault tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proc. IEEE,* vol. 74, pp. 732–741, May 1986.

[15] J-Y. Jou and J. A. Abraham, "Fault tolerant FFT networks," *IEEE Trans. Comput.*, vol. C-37, pp. 548–561, May 1988.

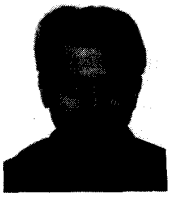
[16] F. T. Luk, "Algorithm-based fault tolerance for parallel matrix equations solvers," in *Proc. SPIE Real Time Signal Proc.*, vol. 564, Aug. 1985, pp. 49–53.

[17] F. T. Luk and H. Park, "An analysis of algorithm-based fault tolerance techniques," in *Proc. SPIE Adv. Alg. Arch. Signal Proc.*, vol. 696, pp. 222–228, Aug. 1986.

[18] F. T. Luk and H. Park, "Fault tolerant matrix triangularizations on systolic arrays," *IEEE Trans. Comput.*, vol. C-37, pp. 1434–1438, Nov. 1988.
[19] V. S. S. Nair and J. A. Abraham, "A model for the analysis of fault tolerant signal processing architectures," in *Proc. 32nd Int. Tech. Symp. SPIE,* San Diego, Aug. 1988, pp. 246–257.
[20] V. S. S. Nair and J. A. Abraham, "A model for the analysis, design and comparison of fault-tolerant WSI architectures," in *Proc. Workshop on Wafer Scale Integration,* Como, Italy, June 1989.
[21] V. S. S. Nair and J. A. Abraham, "Hierarchical design and analysis of fault-tolerant multiprocessor systems using concurrent error detection," in *Proc. Int. Symp. Fault Tolerant Comput.,* Newcastle-upon-Tyne, U.K., June 1990, pp. 130–137.
[22] P. Raghavan, Lecture notes on randomized algorithms, IBM Tech. Rep. RC15340, T. J. Watson Research Center, Yorktown Heights, NY, Jan. 1990, pp. 51–55.
[23] A. L. N. Reddy and P. Banerjee, "Algorithm-based fault detection for signal processing applications," *IEEE Trans. Comput.,* vol. C-39, pp. 1304–1308, Oct. 1990.
[24] D. J. Rosenkrantz and S. S. Ravi, "Improved upper bounds for algorithm-based fault tolerance," in *Proc. 26th Allerton Conf. Comm. Cont. Comput.,* Allerton, IL, Sept. 1988, pp. 388–397.
[25] D. L. Tao, C. R. P. Hartmann, and Y. S. Chen, "A novel concurrent error detection scheme for FFT networks," in *Proc. Int. Symp. Fault Tolerant Comput.,* Newcastle-upon-Tyne, U.K., June 1990, pp. 114–121.
[26] B. Vinnakota and N. K. Jha, "Diagnosability and diagnosis of algorithm-based fault tolerant systems," accepted for publication in *IEEE Trans. Comput.*
[27] B. Vinnakota and N. K. Jha, "A dependence graph-based approach to the design of algorithm-based fault tolerant systems," in *Proc. Int. Symp. Fault Tolerant Comput.,* Newcastle-upon-Tyne, U.K., June 1990, pp. 122–129.
[28] B. Vinnakota and N. K. Jha, "Design of multiprocessor systems for concurrent error detection and fault diagnosis," in *Proc. Int. Symp. Fault Tolerant Comput.,* Montreal, June 1991.
[29] J. D. Russel and C. R. Kime, "System fault diagnosis: Closure and diagnosability with repair," *IEEE Trans. Comput.,* vol. C-24, pp. 1078–1089, Nov. 1975.
[30] S-J. Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," in *Proc. Int. Symp. Circuits Systems,* San Diego, May 1992.

Ramesh K. Sitaraman received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, India, in 1985, and the M.S. degree in computer science from the University of Maryland, College Park. Since 1988 he has been working on his Ph.D. thesis in the Computer Science Department at Princeton University.

His research interests include fault tolerance in multiprocessor systems, performance analysis of packet routing algorithms, theoretical computer science, and pattern recognition.



Niraj K. Jha (S'85–M'86–SM'93) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1981, the M.S. degree in electrical engineering from S.U.N.Y. at Stony Brook, NY, in 1982, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, in 1985.

He is currently an Associate Professor of electrical engineering at Princeton University. From 1985 to 1987 he was an Assistant Professor of electrical engineering and computer science at the University of Michigan, Ann Arbor. He has served as the Program Chairman of the 1992 Workshop on Fault-Tolerant Parallel and Distributed Systems. He has also served on the program committees of the IEEE International Conference on Computer Design, the IEEE International Symposium on Fault-Tolerant Computing, the IEEE International Symposium on Circuits and Systems, and the International Conference on VLSI Design. He has coauthored a book titled *Testing and Reliable Design of CMOS Circuits* (Kluwer Academic Publishers). He has authored or coauthored more than 80 technical papers. His research interests include digital system testing, fault-tolerant computing, computer-aided design of integrated circuits and parallel processing.

He is the recipient of the AT&T special-purpose grant award and the NEC Preceptorship award.