1993

Reliability Analysis of the Hypercube Architecture.

Sieteng Soh
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9401570

Reliability analysis of the hypercube architecture

Soh, Sieteng, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

RELIABILITY ANALYSIS OF THE HYPERCUBE ARCHITECTURE

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Sieteng Soh
B.S., The University of Wisconsin-Madison, 1986
M.S. in E.E., Louisiana State University, 1989
May 1993

Acknowledgments

I am deeply indebted to Dr. Suresh Rai for teaching me various topics on reliability and guiding me through my research in this field. I thank him for his patience and help in writing this dissertation.

I thank Dr. Jerry Trahan for his help and invaluable discussions and also for serving as member of the examining committee. In particular, I thank him for his help in writing Appendix B.

Appreciation is also extended to Drs. Jorge L. Aravena, Ahmed A. El-Amawy, Subhash C. Kak, and Morteza Naraghi-Pour of the Electrical and Computer Engineering Department, Si-Qing Zheng of the Computer Science Department, and Steven H. Weintraub of the Mathematics Department for serving as members of the examining committee.

I would like to thank Dr. Alan H. Marshak, Chairman of the Electrical and Computer Engineering Department, for providing me with financial support during my study in the Department.

This work is supported in part by the Air Force Office of Scientific Research under grant AFOSR-91-0025 entitled "Multiprocessing Systems: Reliability Modeling and Analysis Using Multimode Components and Dependent Failures."

Last, but not the least, I sincerely express my gratitude to my mother, sisters, brothers, and my late father whose patience, understanding, and support have made this study a reality.

Table of Contents

List of Tables

List of Figures

Abstract

This dissertation presents improved techniques for analyzing network-connected (NCF), 2-connected (2CF), task-based (TBF), and subcube (SF) functionality measures in a hypercube multiprocessor with faulty processing elements (PE) and/or communication elements (CE). These measures help study system-level fault tolerance issues and relate to various application modes in the hypercube. Solutions discussed in the text fall into probabilistic and deterministic models. The probabilistic measure assumes a stochastic graph of the hypercube where PE's and/or CE's may fail with certain probabilities, while the deterministic model considers that some system components are already failed and aims to determine the system functionality. For probabilistic model, MIL-HDBK-217F is used to predict PE and CE failure rates for an Intel iPSC system.

First, a technique called CAREL is presented. A proof of its correctness is included in an appendix. Using the shelling ordering concept, CAREL is shown to solve the exact probabilistic NCF measure for a hypercube in time polynomial in the number of spanning trees. However, this number increases exponentially in the hypercube dimension. This dissertation, then, aims to more efficiently obtain lower and upper bounds on the measures. Algorithms, presented in the text, generate tighter bounds than had been obtained previously and run in time polynomial in the cube dimension. The proposed algorithms for probabilistic 2CF measure consider PE and/or CE failures.

In attempting to evaluate deterministic measures, a hybrid method for fault tolerant broadcasting in the hypercube is proposed. This method combines the favorable features of redundant and non-redundant techniques. A generalized result on the deterministic TBF measure for the hypercube is then described. Two distributed algorithms are proposed to identify the largest operational subcubes in a hypercube C_n with faulty PE's. Method 1, called LOS1, requires a list of faulty components and utilizes the CMB operator of CAREL to solve the problem. In case the number of unavailable nodes (faulty or busy) increases, an alternative distributed approach, called LOS2, processes m available nodes in $O(mn)$ time. The proposed techniques are simple and efficient.

Chapter 1

Introduction

Reliability is a critical function of computer systems in today's computer-dependent world. The consequences of computer failure are of significant economic impact and are sometimes even matters of life and death for military, industry, aerospace, and communications applications. Thus, it is imperative that computer designers and system users understand the advantages and limitations of the state-of-the-art in reliability design. Refer to [73] for the details on existing fault-avoidance, fault detection, and redundancy techniques.

For parallel computation, several architectures are suggested in the literature [51]. One of the popular architectures is a hypercube, also known as the Boolean n-cube [34,35,68,71]. Because of its appealing properties, a hypercube architecture is the topology implemented in several multiprocessor systems such as Intel iPSC, JPL Mark III, Ametek System/14, and nCube 2S. A hypercube multiprocessor system offers node and link symmetry, logarithmic diameter, and an ability to host some widely used interconnection networks such as tree, ring, and linear array. To meet the demands for high performance systems, larger sized multiprocessor systems (MPS's) are built, such as the nCube 2S which offers a maximum of 8,192 custom designed processors for its configuration. As the size of the system increases, it is more likely to be less reliable [30]. Thus, reliability metrics are also important aspects for performance evaluation of large MPS's.

Reliability aspects of a multiprocessor system are generally analyzed for system-level fault tolerance in which the faults to be tolerated are processing element (PE) and/or communication element (CE) failures. When there is a faulty component, the following steps are usually carried out in the system:

1) Fault detection - The faulty component is identified by the system.
2) Reconfiguration - If a fault is detected and a permanent failure is located, the system may be able to reconfigure its components to replace the failed component or to isolate it from the rest of the system.
3) Recovery - When the failed PE or CE is repaired, it must be automatically integrated back into the system.

See [73] for the details on the above three steps.

The system-level reliability measures of a hypercube, which are discussed in the literature, fall into one of the following two categories: probabilistic fault tolerant measure (PFTM) and deterministic fault tolerant measure (DFTM). The probabilistic measure assumes a stochastic graph of the n-cube where PE's and/or CE's fail with certain probabilities. The PFTM also evaluates the probability of the hypercube being functional. Existing methods for evaluating probabilistic measures in general networks are exponential in nature; however, for certain structures, efficient techniques have been obtained. The structural property of a hypercube is utilized to solve reliability measures in this dissertation. Also, polynomial time algorithms to generate tighter lower bounds on PFTM are described. These parameters are computed using PE and/or CE failure probabilities which, in turn, are obtained considering the node structure of a typical existing hypercube, namely the Intel iPSC system.

In the deterministic measure, the minimum number of component failures that can be tolerated without resulting in system failure is taken as the figure of merit

for the system. For example, in real time applications or when repairs are hardly available, such as in phased-mission systems, it is imperative to know how vulnerable the system is to failing components. Notice that a deterministic measure assumes that some PE's and/or CE's in the hypercube have already failed. The DFTM, then, aims to determine the hypercube functionality by using measures involving diameter properties or by the requirement set by the users of the system. The existing techniques solve these problems in time exponential in the size of the cube. As demands for larger hypercube systems are increasing, it is obvious that better approaches to these problems are also needed. This dissertation provides efficient heuristics for the evaluation of deterministic measures in hypercubes and compares the techniques with existing methods to illustrate their advantages.

The remainder of this chapter contains description of salient features of multiprocessor systems and a discusssion of some popular architectures. Some functionality criteria are introduced which are used for system-level reliability models of hypercube architecture. An outline of the dissertation can be found at the end of this chapter.

1.1 Multiprocessor Systems

A multiprocessor system consists of a group of N autonomous PE's interconnected by certain network topology. Figure 1.1 shows an MPS in which each PE has its own local memory. Here, PE's communicate with each other by exchanging messages. A message may have to go through some intermediate PE's in the interconnection network before reaching its destination. This type of MPS has been referred to as a message-based parallel processing system [69,71]. Another type of MPS called shared-memory multiprocessor system (Figure 1.2) has PE's communicating through a globally shared memory. Two or more processors needing to use

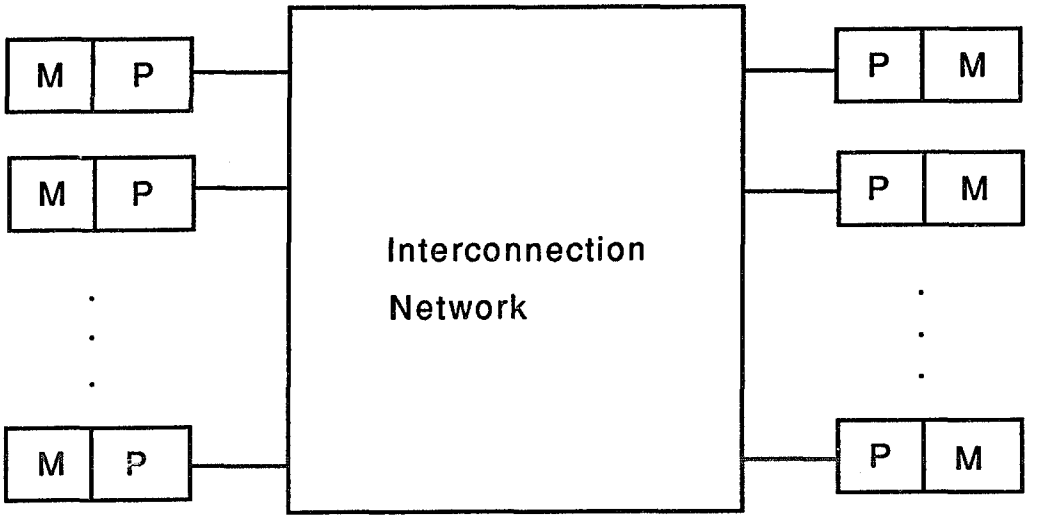


Figure 1.1. Message-based Multiprocessor System

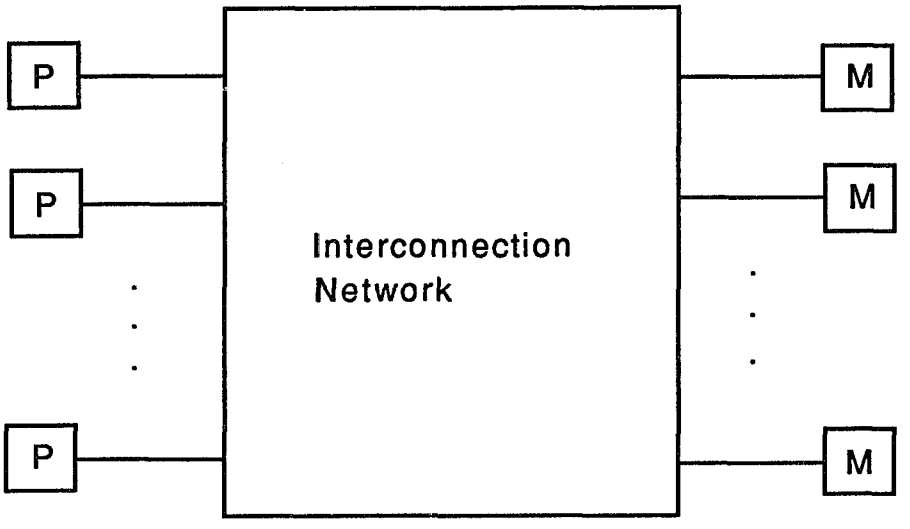


Figure 1.2. Shared-memory Multiprocessor System

the same memory address at the same time should have hardware or software arbitration protocols. Furthermore, memory references are a very large portion of any program's flow, and thus the access time to the global memory has to be kept small to prevent degrading performance and/or speed of the system. This *bottleneck* on the global memory limits the number of processors which can economically be implemented on the shared-memory multiprocessor system.

On the other hand, a message-based multiprocessor system has each of its processors own a local memory unit. For applications in which each processor has most of the data it needs, the number of messages exchanged in the system is relatively small. This allows more processors to be used in the message-based multiprocessor system. Since messages between processors may have to pass through intermediate processors, applications that need extensive intercommunication should be placed in the neighboring processors. This requires that the processors be interconnected in a network topology which, on the average, has a small interprocessor communication time at a reasonable cost. Thus, the network topology (refer to Figures 1.3 through 1.5) for a multiprocessor system plays an important role in determining the performance of the system. If the network does not provide adequate performance, most of the attached processors will frequently be forced to wait for data to arrive. In this dissertation, MPS is used to refer to a message-based multiprocessor system.

A single shared bus shown in Figure 1.3 provides the simplest network topology. However, this topology is inadequate for large multiprocessor systems since the bus unit would be the *bottleneck* of the system as the number of interprocessor communications increases. Figure 1.4 shows a fully connected network topology for $N = 6$. This topology offers the fastest interprocessor communication for the multiprocessor system since each processor is directly connected to every other

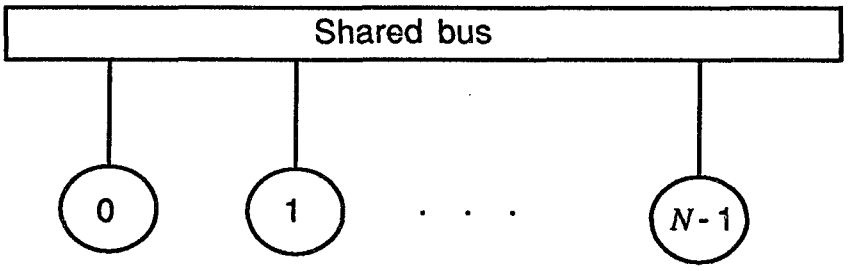


Figure 1.3. A Shared Bus Connecting N Processors

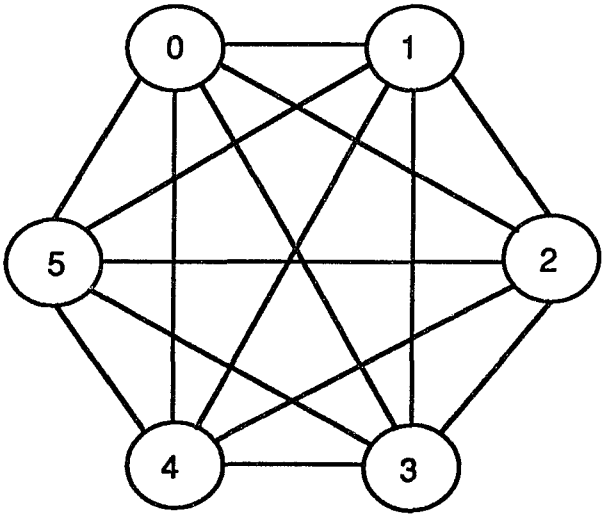
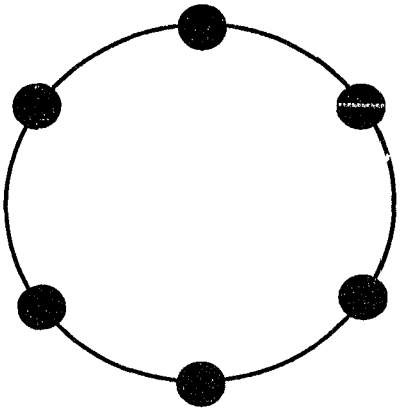
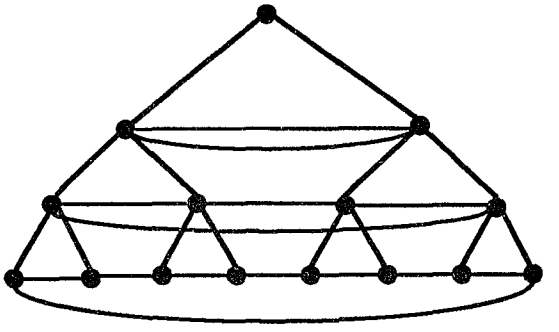


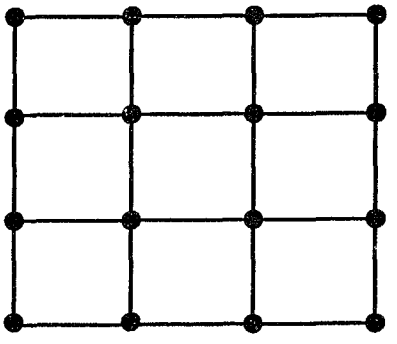
Figure 1.4. A Fully Connected Network with $N = 6$



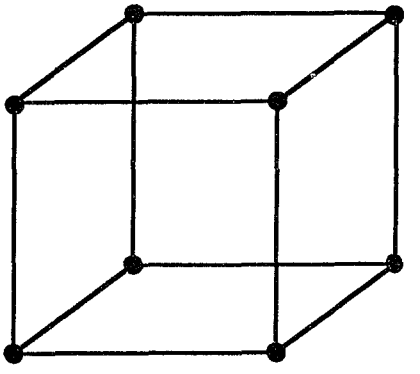
(a) Simple Ring



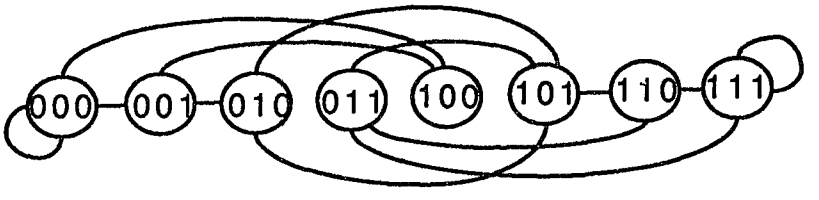
(b) X-tree



(c) Mesh



(d) 3-cube



(e) De Bruijn Network

Figure 1.5. Alternative Topologies for Multiprocessor Systems

processor. However, the topology is impractical for large N since each PE must have N-1 input/output ports. Thus, an N node multiprocessor system interconnected in a fully connected network topology would require $O(N^2)$ communication elements. Several network topologies such as ring, X-tree, mesh, hypercube, and De Bruijn graph offer alternatives for the shared bus and fully connected network topologies (refer to Figure 1.5). Mesh, hypercube, and De Bruijn architectures are useful for solving various classes of problems because they embed many popular topologies such as ring, array, tree, etc. The interprocessor communication time through these topologies is faster than the time in the shared bus, but not as fast as that in the fully connected network. On the other hand, for the N processing elements connected to any of these topologies, each needs only up to $\log_2 N$ communication elements, and thus the overall cost is $O(N \log_2 N)$ communication elements.

1.2 Performance and Reliability Models

Several performance parameters of a multiprocessor system such as connectivity, diameter, mean internode distance, communication link visit ratios, and network expansion increment have been discussed in the literature [13,69]. Connectivity measures the network's resilience and its ability to continue operation despite some component failures. Network diameter, which is also referred to as a maximum internode distance, gives the lower bound on the delay required to propagate information throughout the network. The mean internode distance, on the other hand, is the expected number of nodes a typical message will need to reach its destination. This paramater depends on the message routing distribution that provides the probability that different network nodes exchange messages, which, in turn, depends on the communication requirements of the application and system programs as well as the mapping of these programs onto the network. While the

communication link visit ratio helps locating bottleneck devices in a network that limit the network performance, the network expansion increment gives a measure for the network's expandability with additional nodes.

As the size and complexity of a system increases, the greater number of components makes the failure of some components increasing likely. Consequently, reliability analysis increases in importance to determine if the system can maintain a desired functionality and needs to be included in the performance analysis of the system. Reliability models use several functionality criteria taken as figures of merit for the system. The functionality of a system can be maintained by using system-wide redundancy and by system reconfiguration. The following functionality measures are generally computed for a hypercube system [2,3,9,15,20,21,27,50,55].

1) 2-connected functionality (2CF): The system works as long as two specified nodes in the system are working and connected.
2) Network-connected functionality (NCF): The system works as long as all nodes in the system are working and connected.
3) K-connected functionality (KCF): The system works as long as all nodes in a set ζ of nodes are working and connected, where $|\zeta| = K$.
4) Task-based functionality (TBF): The system works as long as some minimum number of connected nodes are available on the system for task execution.
5) Subcube functionality (SF): The system works as long as some functional minimum degree subcube exists.

In the hypercube structures, the 2CF, NCF, and KCF measures are useful for packet-switching applications because they verify the sturdiness of the topology and depict the probability of succesful flooding (for route setup or packet transmission). The TBF and SF measures incorporate graceful degradation into reliability metrics.

The TBF model is particularly applicable to a large scale multiprocessor system that is used to execute concurrent programs that are insensitive to infrequent changes in the topology of the system. Measure 5) is important because many hypercube algorithms can be executed on various sizes of hypercubes by setting parameters approriately [51,59]. Largest operational subcube (LOS) deals with the identification of the largest size operational subcube so that the system can be reconfigured accordingly. Note, LOS is an extension of a deterministic case of SF. Table 1.1 summarizes the applications of the measures for the probabilistic and deterministic models.

1.3 Layout of the Dissertation

The dissertation addresses efficient algorithms for analyzing both probabilistic and deterministic measures with 2CF, NCF, KCF, SF, and TBF criteria. Chapter 2 provides background material needed for later chapters. Section 2.1 presents the properties of the hypercube architecture including definitions, notations, and assumptions. Section 2.2 includes the description of the node structures of some existing hypercube multiprocessor systems such as Intel iPSC, JPL Mark III, and Ametek's System/14. The section uses the MIL-HDBK-217F to predict the component failure rates for the node structure of the Intel iPSC system.

In Chapter 3, the probabilistic fault tolerant measures for hypercube systems are discussed. First, the literature on the topic is reviewed and then an efficient Computer Aided Reliability EvaLuator (CAREL), which obtains an exact reliability measure, is presented. The notion of shelling orderings for the hypercube topology is also discussed in this chapter. However, CAREL is not advisable for analyzing the reliability measures of large hypercubes because two solution approaches, one using a basic path concept and the other considering preprocessing based on

Table 1.1
Reliability Measures for the Hypercube

Measures	Probabilistic Model	Deterministic Model
2CF	Terminal reliability	Reliable routing [1]
NCF	Network reliability	Reliable broadcasting, gossiping [1]
KCF	K-terminal reliability [1]	Reliable broadcasting, multicasting [1]
TBF	Task-based reliability	-
SF	Functional subcube reliability [1]	Largest operational subcube

[1] Measures not discussed in the text.

shelling, do not help generate results efficiently. The computational complexity of the approach using CAREL is, although, polynomial in the order of the number of spanning trees of the n-cube, is still exponential in the order of the dimension of the cube. Keeping this in view, we aim to obtain lower and upper bounds on the probabilistic measures with the lower bound being of greater interest since it shows that the system is at least this reliable.

Chapter 4 proposes polynomial-time algorithms utilizing the structural properties of the hypercube to obtain tighter bounds on its probabilistic fault tolerant measures. Our NCF and 2CF bounds result in significant improvements over the results obtained using previous techniques. Our proposed algorithms for the 2CF model consider PE and/or CE failure cases. The chapter also includes a method to compute the upper bounds on TBF measure.

Chapter 5 includes a discussion on the deterministic fault tolerant measures for the n-cube, C_n, systems which begins by focusing on the fault tolerant broadcasting, a problem that belongs to the deterministic NCF or KCF model. Next, the discussion focuses on a fault tolerant broadcasting algorithm that combines the favorable features of the existing methods. We show that when failed components do not isolate any C_1 or its subsets, a C_n tolerates up to $3n-6$ node failures. This chapter also describes improved approaches for solving the deterministic model with SF measures. We propose two efficient distributed algorithms to identify largest operational subcubes (LOS's) in a hypercube having failed PE's. The distributed algorithms contrast with the centralized approaches in which the algorithms run in a host / manager node of the hypercube. One of the proposed algorithms, LOS1, requires as its input a list of failed components, while the other proposed algorithm, LOS2, needs a list of available components as its input. For an n-cube, procedure LOS1 is empirically shown to run in polynomial time when only n nodes fail. This

method uses CAREL operators to obtain faster computational time than that given by previous techniques. Algorithm LOS2 utilizes the properties of the hypercube to ignore some components which will not be included in the available largest operational subcube.

Finally, Chapter 6 concludes the dissertation with some suggestions for future work. The appendices provide the proofs of correctness for CAREL and Lemmas 4.1 and 4.2.

Chapter 2

Hypercube Node Reliability

This chapter provides background, notations, definitions, and assumptions that are used throughout the dissertation. It also describes the node structures of some existing commercial hypercube systems, namely Intel iPSC, Ametek System/14, and JPL Mark-III. Several different structures of hypercube architecture such as folded hypercube, cube connected cycles, and generalized hypercube are discussed in the literature [13,47,57], but we concentrate on the most common hypercube structure, the binary n-cube. For system-level reliability models, it is important to know the failure rates of the components. As an example, we consider the Intel iPSC system [68,69] and use MIL-HDBK-217F [19,52] to compute the component failure rates. The handbook gives consistent and uniform information for reliability predictions during the design phase of electronic systems. Reference [52] provides extensive discussion on the role of reliability prediction for reliability modeling, and also discusses part count and part stress methods to help calculate component level failure rates. The part count analysis requires only limited information, however, it should be used only during bid proposal and early design stages when sufficient information is not available for practicing the part stress method. The part stress method is applicable when most of the design is completed and a detailed parts list, including its part stresses, is available. The dissertation uses the part stress method.

2.1 Background

A hypercube of dimension n, C_n, is a graph with 2^n nodes (PE's) and $n \cdot 2^{n-1}$ links (CE's), where each node is labeled with n-bit binary string such that two nodes whose labels differ in exactly one bit position have a link connecting them. In this dissertation, the terms node (link) and PE (CE) are used interchangeably. A C_n is also defined recursively in terms of the graph product operation **X** as [22]:

$$C_n = C_1 \mathbf{X} C_{n-1}.$$

Figure 2.1 illustrates the n-cube topology for $n < 4$. References [32,70] discuss the topological properties of the hypercube. Consider that each node has labels $b_{n-1}b_{n-2} \cdots b_0$, where $b_i \in \{0, 1\}$, and $i \in \{0, 1, \cdots, n-1\}$ refers to the ith dimension of the cube. We use $H(v, w)$ to denote the Hamming distance [53] of the processors v and w. The antipodal of a node v in an n-cube is a node u which is located farthest from v, i.e., $H(v, u) = n$.

Let $G = (V, E)$ represent a stochastic graph model for C_n, where V denotes the set of nodes, and E defines the set of edges/links in the C_n. The degree of a node v is given as the number of its neighbors. Each node in a C_n has degree n. The distance between nodes v and w is the length of a shortest path between the two nodes. The diameter of a network G is defined as the maximum distance over all the pairs of nodes in the network; a C_n has diameter n. A subcube is a subset of a hypercube which preserves the properties of the hypercube.

An *(s,t)-path* is a set of nodes and links in G that provides a connection from a given source node s to a given terminal node t in G. The (s,t)-path is minimal when any proper subset of it does not result in a path between s and t. It is also termed as simple path. On the other hand, a *basic* path between s and t is defined as a path in which two nodes that are not adjacent in the path are not adjacent in

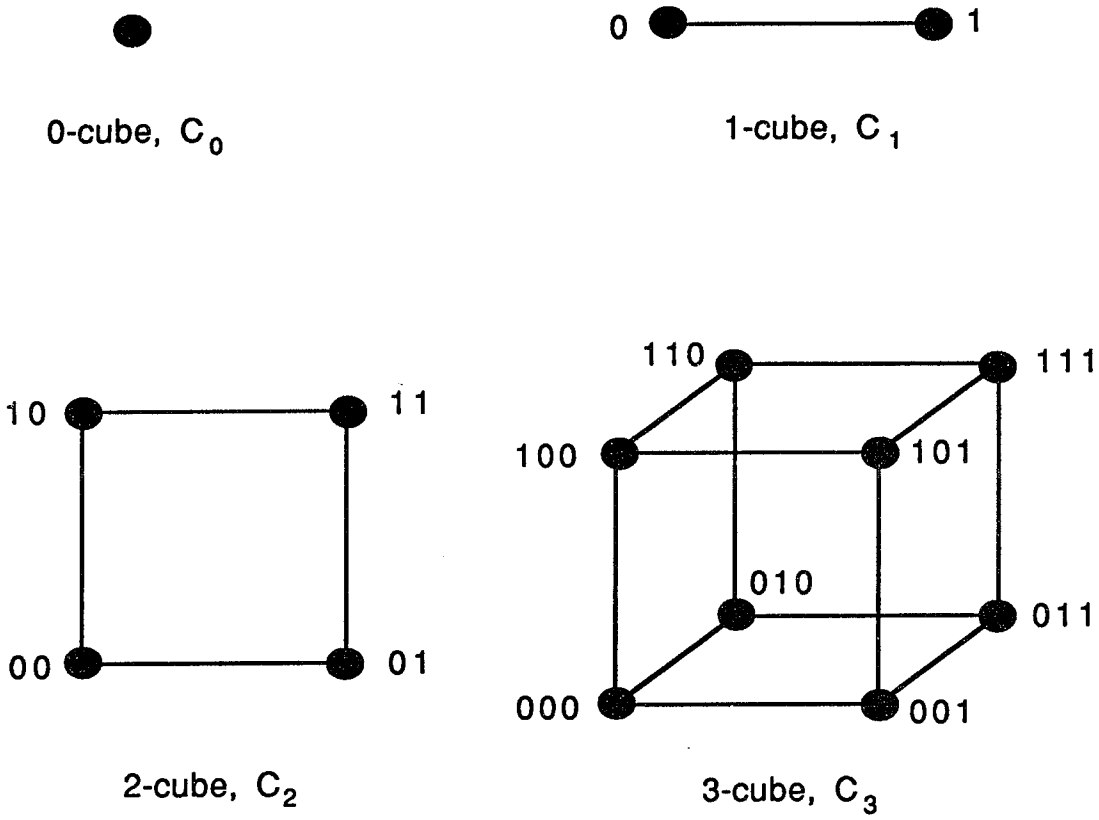


Figure 2.1. Hypercube Topology with Dimension n < 4

G. Note, set of basic paths is a subset of all minimal paths. Without loss of generality, let the addresses of s and t be 0 and 2^n-1, respectively in C_n. These nodes are at diameter distance n. There are $n!$ simple paths in a C_n, each of which traverses n links ($n+1$ nodes). Of the $n!$ paths, n of these are nodes and links disjoint. An *all-path* is defined as a set of nodes and links in G that provides a path between every pair of nodes in G. Thus, a *minimal* all-path is a spanning tree. References [32,88] give the total number of spanning trees, $ST(n)$, in a C_n as

$$ST(n) = 2^{-n}\prod_{j=1}^{n}(2j)^{\binom{n}{j}}. \tag{2.1}$$

A cut is a disconnecting set. A minimal cut (all-cut) corresponds to a minimal set of links in G whose failure ensures that no communication exists between a given (every) s,t node pair.

Assumptions

a) Nodes (links) are statistically identical and have the same failure rate $\gamma(t)$ ($\lambda(t)$).

b) Node (link) failures are statistically independent and exponentially distributed. One can compute node reliability $r(t)$ and link reliability $p(t)$ as functions of time as: $r(t) = \exp(-[\int_0^t \gamma(\tau)d\tau\,])$ and $p(t) = \exp(-[\int_0^t \lambda(\tau)d\tau\,])$. For constant $\gamma(t)$ ($\lambda(t)$), $r = \exp(-\gamma t)$ ($p = \exp(-\lambda t)$). The independence assumption makes the reliability problem mathematically tractable.

c) Each node (link) has two states: good (up) or bad (down).

2.2 Reliability Prediction

To help determine the PE and CE reliabilities, one may use *Intel Components Quality and Reliability* [39], the *Bellcore*'s ARPP (Automated Reliability Prediction

Procedure) [10], or the *AT&T Reliability Manual* [44]. In this dissertation, a reliability prediction method is presented based on the military standard handbook [19,52]. The handbook provides a mathematical framework for electronic component (IC) failure rate derived from the Arrhenius function. This function is used to compute predicted failure rates for IC components. This model has been modified several times based on field data in subsequent releases of the handbook. But for our discussion, we use the latest release on the standard, the MIL-HDBK-217F [52], which includes several multiplicative adjustment factors reflecting device technology, packaging, screening / testing level, environment, operating voltage, process / design maturity, etc. An Arrhenius function applicable for microcircuits, gate / logic arrays, and microprocessors is given as [52]:

$$\lambda_d = 1000 * (K_1\Pi_T + K_2\Pi_E)\Pi_Q\Pi_L \quad \text{FITS}, \tag{2.2}$$

where λ_d represents the device failure rate, K_1 is the circuit complexity factor based on bit count, Π_T is the temperature acceleration factor based on technology, K_2 is the package complexity factor, Π_E is the application environment factor, Π_Q is the screening / testing level factor, Π_L is the device learning (process / design maturity) factor, and the unit FITS represents part failures per million per thousand hours. In what follows, we apply this reliability prediction method to compute the failure rates for PE and CE of the Intel iPSC system.

2.2.1 Node and Link Failures

Figure 2.2 shows a node model in a multiprocessor system. Each node has a processing unit (PU), a memory unit (MU), and a hardware support for the inter-node communication unit (CU). Table 2.1 provides the group of components that belong to MU, PU, and CU for an Intel iPSC, an Ametek System/14, and a JPL Mark-III hypercube system. Figure 2.3 shows an example of a node architecture for

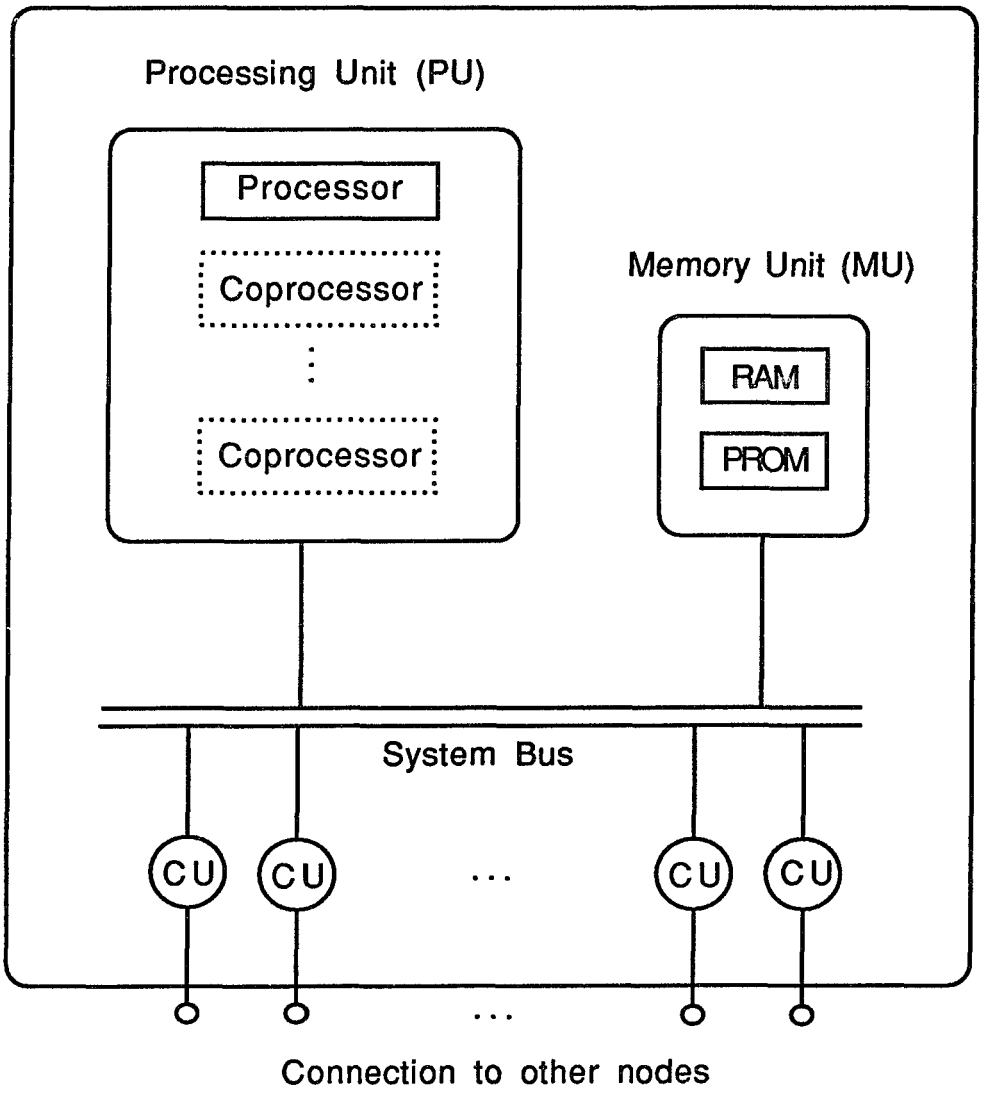


Figure 2.2. Structure of a Node in Hypercube

Table 2.1
A PE Model for Some Existing Hypercube Systems

Systems	Memory Unit	Processing Unit	Communication Unit
Intel iPSC	64K PROM 512K DRAM	Intel 80286 Intel 80287 Intel 82586	Intel 82586
Ametek/14	16K PROM 1M DRAM	Intel 80286 Intel 80287 Intel 80186	2 Intel 8237
JPL Mark-III	4M DRAM	2 M-68020 M-68881	8 bit link drivers FIFO buffers

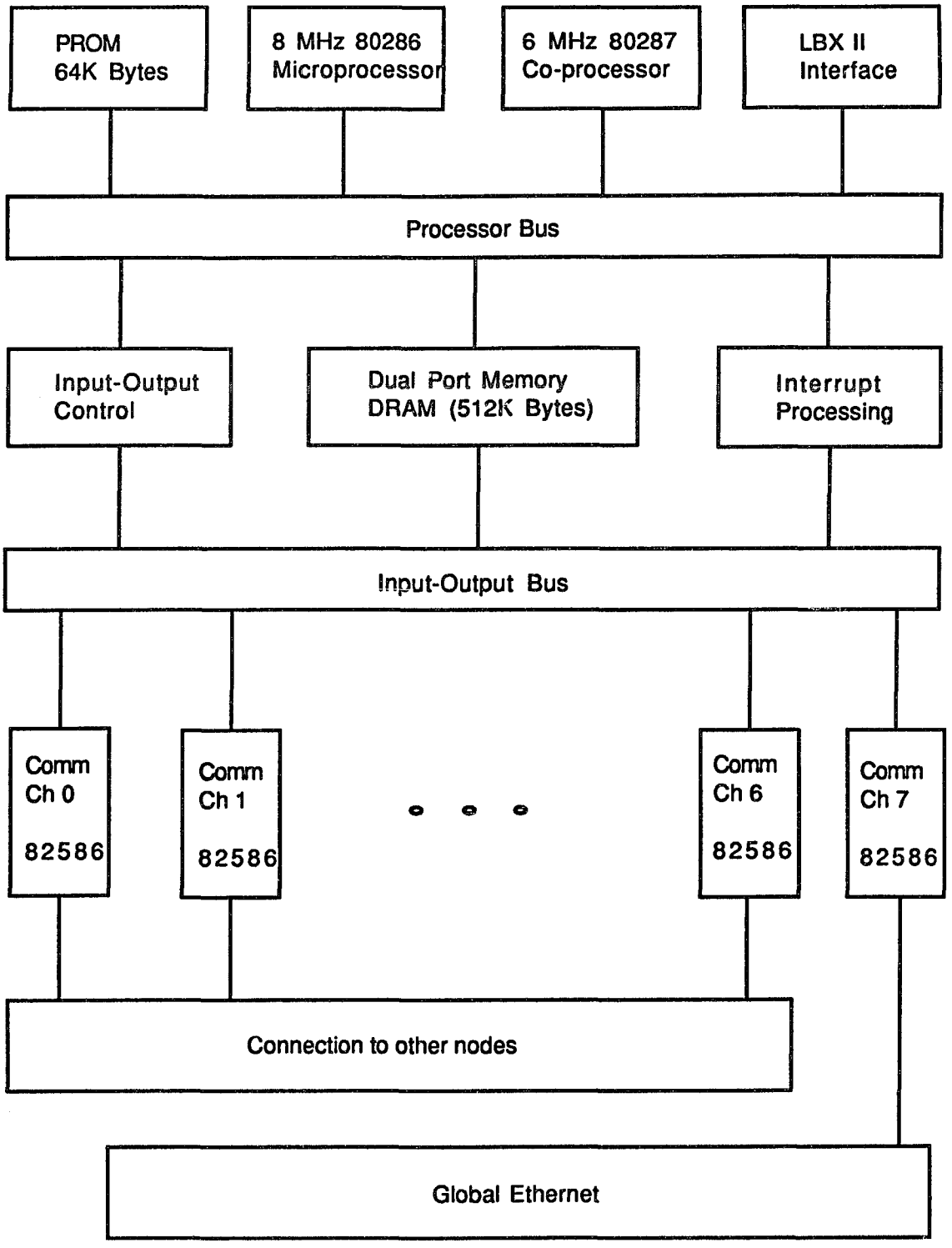


Figure 2.3. Intel iPSC Node Organization.

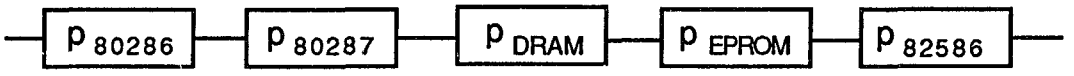
the Intel iPSC [34,35] hypercube. Each node in the Intel iPSC contains a microprocessor (Intel 80286), a floating point coprocessor (Intel 80287), 512K bytes of dual-ported DRAM, 64K bytes of PROM, n Ethernet controllers (Intel 82586), and one additional Intel 82586 used for I/O purposes. Notice that each controllers 82586 is connected to its respective neighbor. Any failure in the controller chip destroys the communication path from the node to the corresponding neighboring node. For our case, link (CE) reliability is computed from a series structure consisting of two Intel 82586 chips and the physical link between them. Figure 2.4a depicts a reliability block diagram illustrating these concepts. Let failures be statistically independent and p_x represent the reliability of a component x. The PE reliability parameter is obtained from the failures in the MU, the last Intel 82586 (I/O) chip, and 80286 and 80287 processor chips. A reliability block diagram to compute PE vulnerability is given in Figure 2.4b.

2.2.2 Failure Rate Computation

In example 2.1, we use Equation (2.2) to predict failure rates for the components used in the Intel iPSC computer system, i.e., the Intel 80286, Intel 80287, DRAM, PROM, and the Ethernet controller 82586. Example 2.2, on the other hand, shows how to predict the failure rates for the PE and CE of the Intel hypercube system.

Example 2.1. Table 2.2 shows the failure rate calculation for an Intel 80286 microprocessor using Equation (2.2). We assume that the system is used in an air conditioned, non-mobile, benign environment such as a computer laboratory or office. We obtain the package thermal specifications for the Intel chips from references [37,38]. Using Equation (2.2), we compute the failure rate for Intel 80286 as

(a)



(b)

Figure 2.4. Reliability Block Diagram
for (a) CE , (b) PE in Intel iPSC System

Table 2.2
Failure Rate Calculation for Intel 80286 Microprocessor

Parameter	Value	Specifications
K_1	0.28	16-bit HMOS III microprocessor
K_2	0.027	68 active pins
Π_L	1	$\geq$ 2 years in production
Π_Q	10	unknown screening levels
Π_E	0.5	nonmobile, temperature and humidity controlled environment.
Π_T	0.35	$T_J = T_C + P * \theta_{JC}$ θ_{JC} = 5.5 C/W (68 lead PGA). Power dissipation = 3.3 Watts (Max). $T_C = 35^o$ C (environment as in Π_E).

$$\lambda_{80286} = 1000 * (0.28*0.35 + 0.027*0.5) * 10 * 1 = 1115 \text{ FITS.}$$

By similar calculations, we obtain the failure rates for a 256K x 1 DRAM chip, $\lambda_{DRAM} = 49.5$ FITS, and the one for a 64K x 8 PROM chip, $\lambda_{EPROM} = 99.3$ FITS. Considering the failure rates for Intel 80287 coprocessor chip and Intel Ethernet 82586 chip as $\lambda_{80287} = 916$ FITS and $\lambda_{82586} = 383$ FITS, respectively, we are able to compute the failure rates for the PE and CE of the Intel iPSC.

Example 2.2. A PE of the Intel iPSC system consists of 64K bytes EPROM, 512K bytes DRAM, an Intel 80286, an Intel 80287, and an Intel 82586 (see Table 2.1). For the memory units, an Intel EPROM 27C512 and 18 - 256K x 1 Intel DRAM 27256 chips, forming the 512K bytes RAM, are assumed to be used. We consider that the system uses 1 parity bit for each byte of its memory units. Assuming exponential distribution for failure rates, we obtain the reliability of the Intel 80286, the Intel 80287, the 512K bytes RAM, the 64K bytes EPROM, and the Intel 82586 chips. Following the reliability block diagram for a PE shown in Figure 2.4b, the reliability of a PE is given as $p_{PE} = e^{-3.4T}$, where T is time in billion hours. Similarly, we compute the reliability of the CE of the system to be $p_{CE} = e^{-0.77T}$.

The PE and CE require logic circuits to combine the PE and CE components. We have excluded the failure rate factors of these circuits from reliability expressions for the PE and CE. We can, though, include these failure rates either by considering them as a function of time t or by using a multiplicative factor incorporated in the final results of the p_{PE} and p_{CE}. In this dissertation, we have used the latter approach. However, in either case, the circuit failure rate for a PE should be larger than that of a CE since a PE uses more complex circuits than does a CE. Let c_1 and c_2 be the circuit reliability parameters for the PE and CE, respectively, for $c_1 < c_2 \leq 1$. Thus, the reliability expressions for the PE and CE are given as $p_{PE} =$

$c_1\, e^{-3.4T}$ and $p_{CE} = c_2\, e^{-0.77T}$, respectively. Table 2.3 illustrates the reliability predictions for the PE and CE for t from 1000 hours to 100,000 hours, and with c_1 = 0.95 and c_2 = 0.99, respectively.

Table 2.3
Component Reliabilities for Intel iPSC

Time (hours)	p_{PE}	p_{CE}
1000	0.94678	0.98924
5000	0.93398	0.98620
10000	0.91824	0.98241
20000	0.88755	0.97487
30000	0.85788	0.96739
40000	0.82920	0.95997
50000	0.80148	0.95261
60000	0.77469	0.94530
70000	0.74879	0.93805
80000	0.72376	0.93086
90000	0.69957	0.92372
100000	0.67618	0.91663

Chapter 3

Probabilistic Fault Tolerant Measures

Probabilistic fault tolerant measures (PFTM) for the hypercube multiprocessor system, addressed in the literature [2,15,42,47,54], are useful for packet-switching applications because they verify the sturdiness of the topology and depict the probability of succesful flooding (for route setup or packet transmission). We discuss exact evaluation methodologies for the measures with NCF, 2CF, and SF criteria.

An exact probabilistic measure for the hypercube is computed by, first, enumerating the success (failure) terms of the cube. Each success (failure) term is treated as a product term of Boolean variables, and, thus, the set of success (failure) terms is represented as a sum-of-products expression. Next, we use a sum-of-disjoint-product (SDP) technique [20,60] to transform the sum-of-product expression into an analogous reliability expression. This chapter presents an efficient SDP algorithm called Computer Aided Reliability EvaLuator (CAREL) [74]. CAREL has been succesfully utilized to evaluate reliability performances of general distributed networks [63,74,75,76]. It is shown that the technique outperforms existing sum-of-disjoint-products methods in terms of solution time [74]. We apply the algorithm to obtain exact values for the probabilistic model with NCF, 2CF, and SF measures for the hypercube.

In our analysis, it is assumed that node and link failures are statistically independent. This assumption is useful since the reliability problem is

mathematically intractable. Nevertheless, for hypercubes, this assumption is justifiable since each node (link) in the system is controlled by different and independent processors (controllers). [Refer to Section 2.2.1]. With perfect links and failing nodes, the computation for NCF measure is straightforward; it is just the product of node reliabilities. Since link failures are assumed to be independent from node failures, the NCF metric for the case when both links and nodes can fail is computed by taking the product of the measures for the separate link and node failure cases. This chapter discusses NCF measure only for the link failure assumption. The 2CF measure is also considered for link failure case because the 2CF measure with node failure case can be solved similarly. The chapter, however, considers the SF model for the node failure case.

The layout of the chapter is as follows: We, first, provide background material on the SDP techniques. To help discuss CAREL we, then, introduce four operators: COM, RED, CMB, and GEN. Some examples for solving the probabilistic model with different functionality criteria are illustrated. The chapter also discusses the shelling property [8] of spanning trees in the hypercube and shows that CAREL evaluates the probabilistic NCF for a hypercube in time polynomial in the order of the number of spanning trees in the cube. We have also included a proof for the correctness of CAREL in Appendix A.

3.1 Background

Several algorithms for exact computation of network reliability are proposed in the literature. These methods are classified as state enumeration, decomposition, inclusion-exclusion, factoring, and sum-of-disjoint products techniques. State enumeration generates the favorable events by examining all possible states and is the simplest method. It is not practical for large networks, however, since the

number of states to be examined increases rapidly. The decomposition technique divides a network into smaller networks whose reliability can be directly computed, but algorithms based on this approach do not produce compact reliability expressions. Both inclusion-exclusion and factoring techniques have similar disadvantages for evaluating large networks. References [20,62] provide extensive discussions on these methods, including their relative merits and demerits.

The sum-of-disjoint-products technique utilizes Boolean concepts to convert a sum-of-product expression for minimal paths or minimal cuts into an equivalent expression of exclusive and mutually disjoint (e.m.d.) terms or an SDP expression [20]. In this form, a logical success (failure) state of a component x is replaced by component reliability (unreliability), and the Boolean sum (product) is replaced by the arithmetic sum (product) to generate the reliability. In other words, the SDP expression is interpreted directly as an equivalent probability expression for network reliability. A drawback of most algorithms based on the manipulation of Boolean sum-of-products or implicants is in the iterative applications of certain operations. The fact that the Boolean function changes at every step may make the method clumsy. Moreover, the algorithms simplify Boolean function using absorption rules [53] and, thus, require a considerable computational effort [31]. Therefore, most SDP techniques are applicable only to small or moderate sized networks. However, with a sharp ($) operator, Grnarov *et al.* [31] present an SDP technique, PROB, which avoids the previously mentioned drawbacks. In the algorithm PROB, a minimal path is represented by a particular binary string (a path identifier), thus, only logical operations need to be performed. Furthermore, the algorithm utilizes the multiple variable inversion (MVI) concept, discussed in Section 3.2, to produce a concise resultant for the final SDP expression.

To further help reduce the overall e.m.d. terms, the minimal paths or minimal cuts are preprocessed (ordered) using some preprocessing techniques. Reference [75] considers cardinality- , lexicographic - , and Hamming distance- ordering methods, and their combinations; it also compares the effect of preprocessing on the performance of the SDP technique with respect to the overall e.m.d. terms generated by the algorithm. Experimentally, the reference shows that cardinality-based ordering is worthwhile doing since it significantly reduces the number of disjoint terms generated (compared to that using a randomly ordered input) and, hence, reduces the CPU time. In the following, our analysis uses minimal paths, however, this is not a limitation since a similar analysis may be presented with minimal cuts. We assume that the minimal paths or minimal cuts of the networks are already available. The minimal (s,t)-paths can be derived by algorithms such as the one proposed by Tarjan [81]. For minimal (s,t)-cuts, reference [84] provides an algorithm that enumerates all minimal (s,t)-cuts of a network in linear time per minimal cut, and references [61,72] discuss methods to obtain the minimal cuts from the minimal paths.

Let F_i denote the event that minimal path P_i is operational. The event F that at least one minimal path works is given by

$$F = \bigcup_{i=1}^{m} F_i, \tag{3.1}$$

where m denotes the number of minimal paths in the network. Let B_i be the Boolean product term corresponding to F_i, that is, the product of Boolean variables corresponding to each component in P_i. Then

$$B = \sum_{i=1}^{m} B_i \tag{3.2}$$

is the Boolean expression corresponding to F. Equation (3.2) is modified either

canonically or conservatively to generate the equivalent SDP expression, $B(disjoint)$. The conservative modification is usually preferred since it is more efficient than canonical modification in which, for l components in the network, 2^l terms are required to determine disjoint terms in B. Most methods consider the single-variable inversion (SVI, discussed in Section 3.2) concept and generate equivalent e.m.d. terms for Equation (3.2) as

$$B_1 + B_2\,\overline{B_1} + B_3\,\overline{B_1}\,\overline{B_2} + \cdots + B_m\,\overline{B_1}\,\overline{B_2}\cdots\overline{B_{m-1}}, \qquad (3.3)$$

where $\overline{B_i}$ is a Boolean product term corresponding to the event that P_i is not operational. The ith term $B_i\,\overline{B_1}\,\overline{B_2}\cdots\overline{B_{i-1}}$ can be evaluated using conditional probability and standard Boolean operations as

$$\mathbf{Pr}(F_i)\cdot\mathbf{Pr}(\overline{F_1}\,\overline{F_2}\cdots\overline{F_{i-1}}\,|\,F_i) = \mathbf{Pr}(F_i)\cdot\mathbf{Pr}(E_1\,E_2\cdots E_{i-1}), \qquad (3.4)$$

where $\overline{F_j}$ denotes the event that minimal path P_j is down, and E_j represents a conditional event $\overline{F_j}\,|\,F_i$ [53] and defines an event that P_j is down given that minimal path P_i is operational. The probability, $\mathbf{Pr}(F_i)$, can be directly computed as the product of the reliabilities of nodes and/or links in path P_i since component failures are assumed to be statistically independent. The second term of Equation (3.4) is solves as $\mathbf{Pr}(E_1\,E_2\cdots E_{i-1}) = \prod_{j=1}^{i-1}\mathbf{Pr}\,(E_j)$ if the E_j's represent independent events. However, the E_j's are, in general, not independent [20,62]. Computing $\mathbf{Pr}(E_1\,E_2\cdots E_{i-1})$ is the most time consuming process in the SDP technique.

Various researchers [20,62] have worked on this philosophy and have proposed methods to generate disjoint expressions for $(F_i,\ F_j)$ pairs, and also to solve $\mathbf{Pr}(E_1\,E_2\cdots E_{i-1})$ in Equation (3.4). The following three propositions, P_I through P_III, convert F into $F(disjoint)$. These propositions state the basic principles behind most SDP methods in the literature.

Proposition P_I. Define intermediate term(s) T_i as:

$$T_i = \bigcup_{j=1}^{i-1} F^j \mid_{\textit{each literal of } F_i \rightarrow 1}, \tag{3.5}$$

where $F^1 = F_1$ and $F^i = F_i \; OP_1 \; T_i$. Here, F^i refers to the equivalent e.m.d. term(s) for F_i. The operation 'OP_1' is a necessary disjointing operator. (Table 3.1 lists various operators.) The $F(\textit{disjoint})$ expression is, then, given by

$$F(\textit{disjoint}) = \bigcup_{j} F^j. \tag{3.6}$$

Algorithms in [58,64] and method 1 in [74] make use of proposition P_I.

Proposition P_II. For each term F_i, $1 < i \leq m$, T_i is defined as the union of all predecessor terms $F_1, F_2, \cdots, F_{i-1}$ in which any literal that is present in both F_i and any of the predecessor terms is deleted from those predecessor terms, i.e.,

$$T_i = \bigcup_{j=1}^{i-1} F_j \mid_{\textit{each literal of } F_i \rightarrow 1}. \tag{3.7}$$

Consider $F^1 = F_1$, and define $F^i = F_i \; OP_2 \; T_i$. Equation (3.6), then, obtains the equivalent $F(\textit{disjoint})$ expression. Hariri and Raghavendra [33], Rai and Aggarwal [60], Bennetts [11], and Soh and Rai (with method 2) [74] have based their techniques on proposition P_II.

Proposition P_III. For $1 < j \leq m$, use operation 'OP_3' to perform

$$F^j = (\; \cdots \; ((F_j \; OP_3 \; F_1) \; OP_3 \; F_2) \; OP_3 \; \cdots \;) \; OP_3 \; F_{j-1}. \tag{3.8}$$

Equation (3.8) obtains a set of disjoint terms corresponding to F_j. Note, $F^1 = F_1$, and OP_3 represents an appropriate disjointing operator. The $F(\textit{disjoint})$ expression is given by Equation (3.6). Abraham [1], Grnarov *et al.* [31], and Tiwari and

Table 3.1.
OP_1 through OP_3 Used with Boolean Algebraic Techniques
(++ Proposed Algorithm)

Operator	Function	Reference
OP_1	X-operator	[58]
	Method 1, CMB (*) operator	++
OP_2	Relative complement and Procedure 1	[11]
	Cutset Disjoint Procedure	[33]
	E-operator	[60]
	Method 2, CMB (*) operator	++
OP_3	COMPARE( ) function	[1]
	$ operator	[15]
	Boolean negation	[82]
	Modified $ operator	[85]

Verma [82] have proposed their methods using the P_III concept. $F(disjoint)$ expressions obtained from different propositions should be identical when expanded.

3.2 Existing SDP Techniques - A Comparison

The problem of computing the probability in Equation (3.4) to obtain $F(disjoint)$ of Equation (3.6) is known to be NP-hard [8]. Hence, for comparing different SDP algorithms, researchers have often used parameters such as the number of terms in the resulting SDP expression, the amount of storage needed, and the execution times of programs in computing specific problems. Use of multiple-variable inversion (MVI), in addition to preprocessing the sum of product terms (see Section 3.1), further reduces the overall number of resulting disjoint terms [86]. This concept is illustrated in the example which follows. Typically, an SDP technique uses a complemented Boolean variable, $\overline{x_i}$ (indicating a failed component i), and an uncomplemented variable, x_i (denoting a functional component i). This Boolean representation is called single-variable inversion (SVI). Thus, in the SVI form, a Boolean expression representing five components, with components 2 and 5 in a *don't care* state, component 4 in a functional state, and components 1 or 3 in a failed state, would be represented by an expression $x_4(\overline{x_1} + \overline{x_3})$. On the other hand, an equivalent but more concise Boolean expression for the example is given in MVI form as $x_4\overline{x_1 x_3}$. Reference [86] reviews and discusses the relative merits and demerits of the following four sum-of-disjoint products techniques that have used MVI concept: PROB [31], GKT-VT [85], KDH88 [36], and CAREL [74]. The GKT-VT is efficient compared to PROB [85] and is able to solve large networks; however, it obtains the terminal reliability of a network with 425 minimal (s,t)-paths in 46.3 CPU hours. Obviously, we still need an efficient algorithm (on the application of Boolean algebra) to solve reliability problems for large networks. CAREL [74]

provides a solution in this direction since it computes the terminal reliability parameter for the network of Figure 19 in [74] (with 780 minimal (s,t)-paths) in less than a minute of CPU time. Note, the network used in [85] (with 425 paths) is a reduced form of the network of Figure 19 in [74].

Propositions P_I through P_III maintain the minimal paths or minimal cuts list in memory (Equation (3.1)). Consider a "1" for an operative component and a "0" for *don't care*, and utilize the bit representation technique [4]. The memory requirement is, then, $\lceil l/w \rceil$ words per minimal path (cut), where l is the number of components in the network $G = (V, E)$, and w is the word size. Proposition P_I makes F_i disjoint with respect to $\bigcup_{j=1}^{i-1} F^j$, while propositions P_II and P_III utilize $\bigcup_{j=1}^{i-1} F_j$. Should we have similar operations to implement Equations (3.5) and (3.7), the proposition P_I would require more operations than that needed for P_II. Generally, an F_i generates more than one e.m.d. term for F^i, and, hence, the number of terms involved in $\bigcup_j F^j$ is larger than that in $\bigcup_j F_j$. Note, in proposition P_I, the generated e.m.d. terms must be kept in the memory to implement Equation (3.5) which is not the case for P_II or P_III. This makes proposition P_I sequential. Moreover, P_I demands a huge memory space to evaluate a large network. On the other hand, P_II and P_III have implicit parallelism, making it easier for the programmers to implement them on parallel systems. Overall, the propositions P_II and P_III provide advantages in comparison with P_I.

An analysis of performance comparison between a typical example of propositions P_II and P_III is discussed in [33]. SYREL [33], an implementation technique for the E-operator [60], is shown to have better performance in comparison with the $-operator [31]. It means proposition P_II outperforms P_III. Moreover, proposition P_II offers a faster implementation approach than that in P_I or P_III. The bit

vector implementation of $\bigcup_j F_j$ makes the realization of Equation (3.7) w (word size) times faster than generating Equation (3.5) based on proposition P_I. The following section presents CAREL [74].

3.3 Computer Aided Reliability EvaLuator (CAREL)

CAREL is a sum-of-disjoint products network reliability algorithm. CAREL is more efficient than existing techniques since it uses conditional probability to significantly reduce the number of terms involved in Equation (3.4) [74]. Computing $\Pr(E_1E_2 \cdots E_{i-1})$ in Equation (3.4) is the most time consuming process in the SDP technique, and, hence, fewer terms involved in the process results in reduced CPU time. Furthermore, CAREL utilizes the MVI concept to obtain a more concise expression for the final e.m.d. terms. CAREL obtains a symbolic reliability expression for the network. The resultant network reliability is computed by substituting a logical success (failure) in the expression by a given component reliability (unreliability). Generating a symbolic reliability expression is advantageous for two main reasons. Firstly, many networks have a fixed topology while the reliability of their components changes with time. Thus, reliability reevaluation is simpler with symbolic expressions. Furthermore, the sensitivity to changes in component reliabilities can be readily determined by utilizing symbolic expression. Secondly, in some applications, it is desired to improve the reliability of a network under a given cost constraint. A symbolic reliability expression can be used to help identify the critical components to optimize the reliability. In other words, the symbolic reliability expression can be readily used to help design (synthesize) optimal networks with certain costs and/or reliability constraints. In Section 3.3.1, we discuss a notational concept that is useful to describe CAREL. Section 3.3.2 describes four operators: COM, RED, CMB, and GEN, which form the basis for CAREL.

3.3.1 Notation

For a network $G = (V, E)$, consider a set of minimal paths P_j's. A path identifier F_j identifies P_j in a cubical notation [53] using a string of symbols from {0, 1}. Thus, an operative component in P_j is represented by a "1" in F_j, while a *don't care* state is represented by a "0". An F^j denotes the e.m.d. term(s) and is generated for an F_j. To help obtain F^j, conditional events E_j's (defined later) are utilized. An F^j uses $\{-\beta^I, 0, 1\}$, while an E_j is composed of complemented and absent variables and requires $\{-\beta^I, 0\}$ symbols [64]. The β represents a positive integer, and $I \in \{0, 1, \cdots\}$ is a superscript or index. We use a superscripted negative integer to represent a complemented variable (failed state of a component). Note the following :

a) An uncomplemented, or complemented, or absent variable is replaced by "1", "$-\beta^I$", or "0" in the position of the variable, respectively.

b) A "$-\beta^I$" represents complements of β number of variables which are grouped together, and the index I represents the least significant bit position among the β variables. [a "-1" signifies a single variable complement. Use of index is optional with "-1".]

To help illustrate the concept of this new notation, a term $\overline{x_3 x_2} x_1 \overline{x_0}$ is represented as $(-2^2\ -2^2\ 1\ -1^0)$. Similarly, a product term $\overline{x_6 x_5 x_3}\ \overline{x_4 x_2} x_0$ is denoted as $(-3^3\ -3^3\ -2^2\ -\ 3^3\ -2^2\ 0\ 1\ )$. The need of a superscript or index is illustrated with the example of a Boolean term $\overline{x_5 x_4}\ \overline{x_3 x_2}\ \overline{x_1 x_0}$ represented as $(-2^4\ -2^4\ -2^2\ -2^2\ -2^0\ -2^0)$. If the indices are not used, a notation of (-2 -2 -2 -2 -2 -2) for this example, in all likelihood, would be wrongly interpreted. The advantage of $(-\beta^I, 0, 1)$ notation lies in its uniqueness in handling complemented variables that are grouped together [31,64,85].

3.3.2 CAREL Operators

Given a set of path identifiers $\bigcup_{i=1}^{m} F_i$, we want to generate the e.m.d. terms F^i generated from the F_i for all i. The COM operator generates a set of conditional events E_j, $j = 1, \cdots, i-1$ for an F_i. The conditional event set E_j's describes the events that a minimal path identified by F_i is operational, while minimal paths P_j, for each $j = 1, 2, \cdots, i-1$ fail. The RED operator, on the other hand, is used to remove the redundant conditional event from the generated set E_j's. We call the non-redundant events minimal conditional events (MCE). If we have only one term in the MCE set or if the events are mutually independent among themselves, we generate the disjoint terms F^i directly. But, in general, the MCE's are not mutually independent, and therefore we need the CMB operator to compute $\mathbf{Pr}(E_1 E_2 \cdots E_{i-1})$. Refer to Equation (3.4). Finally, the last operator, GEN, gives the disjoint terms F^i.

COM (\ ) operator: The COM1 (COM2) version follows proposition P_I (P_II). For P_I, consider $F_k = (\, a_{l-1} \cdots a_1 a_0 \,)$ and $F^j = (\, b_{l-1} \cdots b_1 b_0 \,)$, where $a_i \in \{0, 1\}$, and $b_i \in \{ -\beta^I, 0, 1 \}$. The COM1 (\ ) is, then, defined as

$$F_k \setminus F^j = \begin{cases} \varnothing; & \text{if } a_i{=}1 \text{ in all those } \beta \text{ positions where } b_i\text{'s are } -\beta^I. \\ & \text{It shows that } F_k \text{ and } F^j \text{ are mutually disjoint.} \\ E_k & \text{otherwise} \end{cases}$$

where the conditional set E_k is $(\, c_{l-1} \cdots c_1 c_0 \,)$. A c_i $(= a_i \setminus b_i)$ is obtained using following table:

b_i

a_i	$-\beta^I$	0	1
0	0	0	$-\alpha^I$
1	0	0	0

Here, an α represents the total number of places where a (0, 1) pair in (F_k, F^j)

occurs. For example, consider F_k as (0 1 0 1 0 0 1), and F^j as (1 -2^4 -2^4 -2^1 1 -2^1 0). The COM1 operator obtains E_k as given below:

$$\begin{array}{ll} F_k & 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \\ F^j & 1 \;\; -2^4 \;\; -2^4 \;\; -2^1 \;\; 1 \;\; -2^1 \;\; 0 \\ \hline E_k & -2^2 \;\; 0 \;\; 0 \;\; 0 \;\; -2^2 \;\; 0 \;\; 0 \;\; . \end{array}$$

Notice that $\alpha = 2$, as two (0, 1) pairs are present. On the other hand, the (\ ) operation with F^j (1 -2^3 -2^1 -2^3 1 -2^1 0) is shown to be null (∅).

$$\begin{array}{ll} F_k & 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \\ F^j & 1 \;\; -2^3 \;\; -2^1 \;\; -2^3 \;\; 1 \;\; -2^1 \;\; 0 \\ \hline E_k & \text{null set } (\varnothing) \;\; . \end{array}$$

For proposition P_II, the COM (\ ) operator requires the two path identifiers $F_k = (a_{l-1} \cdots a_1 a_0)$ and $F_j = (b_{l-1} \cdots b_1 b_0)$ where both $a_i, b_i \in \{0, 1\}$. The COM2 is defined in a straightforward manner as $E_k' = F_k \setminus F_j = F_k \; pl \; F_j \; df \; F_k$; where *pl* and *df* are set operators. For operands $F_k, F_j \in \{0, 1\}$, the operators *pl* and *df* are bitwise *or* and *xor*, respectively. Let α be the total number of 1's present with E_k'. Replace 1's in E_k' by $-\alpha^I$ to generate the conditional event E_k. For example, consider *pl* and *df* operators for F_k and F_j given below:

$$\begin{array}{lll} F_k & 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 & \\ F_j & 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 1 & \text{; use } pl \text{ (bitwise } or\text{) operation} \\ & 0 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \;\; 1 & \\ F_k & 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 0 & \text{; use } df \text{ (bitwise } xor\text{) operation} \\ E_k' & 0 \;\; 0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; . & \end{array}$$

The conditional event E_k is, then, obtained from E_k' by replacing "1" by -2^I in the positions of "1". Thus, $E_k = 0 \;\; 0 \;\; 0 \;\; 0 \;\; -2^0 \;\; 0 \;\; -2^0$. The COM1 operator detects and eliminates some of the redundant terms while other redundancies are deleted by the RED operator. The COM2 operator does not check redundancy. Nonetheless, the COM2 operator is bit implementable and offers a great advantage over COM1 from the aspects of computer memory and speed.

RED (/) operator: Consider two conditional events $E_j = (c_{l-1} \cdots c_1 c_0)$, and $E_k = (d_{l-1} \cdots d_1 d_0)$, where $c_i, d_i \in \{ -\beta^I, 0 \}$. For $\alpha^J, \delta^K \in \beta^I$, the RED (/) operation is given by

$$E_j/E_k = \begin{cases} E_j; & \text{if } c_i = -\alpha^J \text{ in all those } \alpha \text{ positions where } d_i = -\delta^K \text{ and } \delta > \alpha. \\ E_k; & \text{if } d_i = -\delta^K \text{ in all those } \delta \text{ positions where } c_i = -\alpha^J \text{ and } \alpha > \delta. \\ E_j, E_k; & \text{otherwise.} \end{cases}$$

Observe that, by REDucing either E_j or E_k, we remove redundant events. Henceforth, the remaining non-redundant E_i's are said to form a minimal conditional event (MCE). The following examples illustrate the RED operation:

(i)	E_j	$0\ -2^1\ 0\ -2^1\ 0$	(ii)	E_j	$0\ -2^1\ 0\ -2^1\ 0$
	E_k	$0\ -3^1\ -3^1\ -3^1\ 0$		E_k	$-3^2\ -3^2\ -3^2\ 0\ 0$
	E_j	$0\ -2^1\ 0\ -2^1\ 0$			Retain both E_j and E_k .

For (i), $-\alpha^J = -2^1$, $-\delta^K = -3^1$, and $\delta > \alpha$. Moreover, both -2^1's are in the positions where -3^1's are present. Thus, E_k is redundant, and the result is E_j. A similar explanation follows for (ii). The RED operator, defined above, is suitable for P_I. For notational simplicity, let us call it RED1. With proposition P_II, a simpler version (RED2) is adopted. The COM2 operation generates event E_k' which contains only 0's and 1's. Using E_k''s, the redundancy checking required in RED2 operator is brought down to set theoretic operations *pl* and *df* (see [74]). This observation will help make RED2 implementation faster than that for RED1. Using RED2, the examples (i) and (ii) can be solved as

(i)	E_j'	0 1 0 1 0	
	E_k'	0 1 1 1 0	; *pl* (*or* operation)
		0 1 1 1 0	
	E_k'	0 1 1 1 0	; *df* (*xor* operation)
		0 0 0 0 0	; means E_k' is redundant

(ii)

E_j'	0 1 0 1 0	
E_k'	1 1 1 0 0	; *pl* (*or* operation)
	1 1 1 1 0	
E_k'	1 1 1 0 0	; *df* (*xor* operation)
	0 0 0 1 0	; non null means retain both terms .

CMB (*) operator: The CMB (*) operator processes MCE E_i's as its operands. Before applying CMB, partition the set of E_i's into independent (IG) and dependent (DG) groups. The MCE's that belong to IG are already mutually independent among themselves. Thus, generating the disjoint terms F^i from IG is straightforward. It has been observed in [33] that most minimal (s,t)-paths in large computer networks (which are of the loosely connected type) do not have common elements among themselves. The partitioning of E_i's into IG and DG can be embedded in the implementation of the RED (/) operation (refer to Section 3.4.2), which avoids an unnecessary taxing of the implementation of the CMB (*) operator. Moreover, the processing cost (as will be clear later in this section) for IG is far less than that for DG, and, hence, the overall improvement in the performance of the algorithm is obvious.

Definition. Consider MCE's E_j and E_k whose elements $c_i, d_i \in \{-\beta^I, 0\}$. For $1 \le i \le l$, if there exists at least one (c_i, d_i) pair for $c_i, d_i \neq 0$, the E_j and E_k are said to form a dependent group (DG).

As an example assume E_j as $(-2^4\ 0\ -2^4\ 0\ 0\ 0\ 0)$ and E_k as $(-3^2\ 0\ 0\ -3^2\ -3^2\ 0\ 0)$. Here, E_j and E_k belong to DG since they have a common element in position 6. Use this definition to select DG's from non-redundant MCE E_i's. The remaining E_i terms form IG's. The (E_j, E_k) entry in IG has no common elements among themselves. Thus, the (c_i, d_i) pair will always be of the types (0,0), $(-\alpha^J,0)$, and $(0, -\delta^K)$; therefore, terms such as $(0\ -2^1\ 0\ 0\ 0\ -2^1\ 0)$ and $(0\ 0\ 0\ 0\ 0\ 0\ -1)$ belong to IG. These terms are independent with both E_j and E_k considered above. For notational

simplicity, we denote the elements of independent (dependent) group by IG_i (DG_i). For | MCE | = r, | IG | = η, and | DG | = γ, $\eta + \gamma = r$. An element E_i belongs to either IG or DG. The CMB operator differentiates our algorithm CAREL from [33,60,64]. The following two cases define CMB; **CASE 1** is used for the independent group, and **CASE 2** is used for the dependent group.

CASE 1. [CMB for independent group]. For $1 \leq j < \eta$, the CMB operates iteratively as

$$IG_j \ * \ IG_{j+1} \rightarrow IG_{j+1}, \tag{3.9}$$

where '*' is a *pl* operation such that 0 *pl* 0 = 0, 0 *pl* $-\delta^K = -\delta^K$, and $-\alpha^J$ *pl* 0 = $-\alpha^J$. Equation (3.9) states that we will eventually get one term IG_η.

Example 3.1. Consider four IG_i's: IG_1 (00-2^4-$2^4$0000), IG_2 (-10000000), IG_3 (0-$3^1$00-$3^1$0-$3^1$0), and IG_4 (00000-$2^0$0-2^0). IG_η is derived as follows:

IG_1	0	0	-2^4	-2^4	0	0	0	0
IG_2	-1	0	0	0	0	0	0	0
IG_2	-1	0	-2^4	-2^4	0	0	0	0
IG_3	0	-3^1	0	0	-3^1	0	-3^1	0
IG_3	-1	-3^1	-2^4	-2^4	-3^1	0	-3^1	0
IG_4	0	0	0	0	0	-2^0	0	-2^0
IG_η	-1	-3^1	-2^4	-2^4	-3^1	-2^0	-3^1	-2^0

CASE 2. [CMB for dependent group]. In this case, the CMB operator is quite involved. To define this operator, use Steps 1 and 2 below.

Step 1. DG_1 * DG_2 generates the two terms TG_1 and TG_2. The term TG_1 has $-\theta^I$ in those θ positions where entries in DG_1 and DG_2 are both negative, while the other contents of TG_1 are 0's. TG_2 has '1' in all the θ positions, while its remaining entries are generated from DG_1 and DG_2 using 0 *pl* $x^J = x^J$ *pl* 0 = x^J; $x^J \in \{-\beta^I, 0\}$. Before applying the *pl* operation, update variable x by adding θ to x. The

entries in TG's belong to $\{-\beta^I, 0, 1\}$.

Step 2. Consider $TG_j = (f_{l-1} \cdots f_1 f_0)$ and $DG_i = (d_{l-1} \cdots d_1 d_0)$; where $f_i \in \{-\alpha^J, 0, 1\}$, $d_i \in \{-\delta^K, 0\}$ and $\alpha^J, \delta^K \in \{\beta^I\}$. The following substeps generates TG_i's.

```
for (all DG_i) begin        /* i = 3, ··· */
   for(all TG_j) begin      /* j = 1, ··· */
      call DG (TG_j, DG_i, TG');      /* TG' is the result */
      j = j + 1;
   end;
   TG = TG';          /* we create new TG_j list */
end;
```

Step 2 needs a procedure **DG** to obtain various TG_i's. An algorithm for **DG** (TG_j, DG_i, TG') is as follows :

/* To make the algorithm easier to follow, we provide an example for each *case* (a) to (f) in Boolean expression. Note, X is any Boolean expression */

```
while (true) begin          /* forever, exit only by return */
   delta = Number of (1,-δ^K) pairs;
   alpha = Number of (-α^J,-δ^K) pairs;
   /* Consider the following cases of (delta,alpha) */
   (delta == δ, alpha == don't care) :          /* case (a) */
   begin                    /* (abcX) (abc-bar) = ∅ */
      return;
   end;
   (delta > 0, alpha == don't care) :           /* case (b) */
   begin                    /* (abX)(abcde-bar) = (abX)(cde-bar) */
```

for(k=0 to l−1) **begin**

if($DG_i[k] == -\delta^K$) **begin**

if($TG_j[k] == 1$)

$DG_i[k] = 0$;

else

$DG_i[k] = DG_i[k] + delta$;

end;

end;

end;

(*delta* == 0, *alpha* == α) : /* *case* (c) */

begin /* CMB for independent group */

$TG_j' = TG_j\ pl\ DG_i$; /* $(abX)(\overline{cd}) = ab\ \overline{cd}X$ */

append TG_j' to TG';

return;

end;

(*delta* == 0, *alpha* == α) : /* *case* (d) */

begin /* $(\overline{abc})\ (\overline{ab}X) = \overline{ab}X$ */

append TG_j to TG';

return;

end;

(*delta* == 0, *alpha* == δ) : /* *case* (e) */

begin /* $(\overline{abcX})\ (\overline{ab}) = \overline{ab}X$ */

$TG_j' = DG_i\ pl$ (the rest of elements in TG_j other than $-\alpha^J$);

append TG_j' to TG';

return;

end;

```
OTHERWISE :          /* case (f) */
/* (abeX) (abcd) = abX + (ab eX)(cd)      */
/* note, the first term above is one of the final result */
begin
   for(k=0 to l-1) begin
     if(DG_i[k] == -δ^K) begin
       if(TG_j[k] == -α^J) begin
         TG_j'[k] = -alpha;
         TG_j[k] = 1;
         DG_i[k] = 0;
       end;
       else begin
         TG_j'[k] = TG_j[k];
         DG_i[k] = DG_i[k] + alpha - delta;
       end;
     end;
     else if(TG_j[k] == -α^J) begin
       TG_j'[k] = 0;
       TG_j[k] = TG_j[k] + alpha;
     end;
     else
       TG_j'[k] = TG_j[k];
   end;
   append TG_j' to TG';
  end;
end;
```

Cases (a) through (f) are obtained from Theorems A.1 and A.2 in Appendix A. A proof on the correctness of the CMB operator is also given in the appendix.

Example 3.2. Assume DG_1 ($-3^6 0 -3^6 -3^6 0 0 0 0 0 0$), DG_2 ($-4^0 0 -4^0 0 -4^0 0 0 0 0 -4^0$), DG_3 ($-3^4 0 -3^4 0 0 -3^4 0 0 0 0$) and DG_4 ($-4^3 0 0 -4^3 0 -4^3 -4^3 0 0 0$), where $DG_i \in$ DG for $1 \leq i \leq 4$.

Step 1. We generate the two terms TG_1 and TG_2 as follows :

DG_1	$-3^6\ 0\ -3^6\ -3^6\ 0\ 0\ 0\ 0\ 0\ 0$
DG_2	$-4^0\ 0\ -4^0\ 0\ -4^0\ 0\ 0\ 0\ 0\ -4^0$
TG_1	$-2^7\ 0\ -2^7\ 0\ 0\ 0\ 0\ 0\ 0\ 0$
TG_2	$1\ 0\ 1\ -1\ -2^0\ 0\ 0\ 0\ 0\ -2^0$

Step 2. Consider $TG_1 * DG_3$ and $TG_2 * DG_3$. Using procedure **DG**, $TG_1' = TG_1$ since $TG_1 * DG_3$ is of *case* (d) (refer to the procedure). The $TG_2 * DG_3$ is computed as follows :

TG_2	$1\ 0\ 1\ -1\ -2^0\ 0\ 0\ 0\ 0\ -2^0$	
DG_3	$-3^4\ 0\ -3^4\ 0\ 0\ -3^4\ 0\ 0\ 0\ 0$	; *case* (b), *delta* = 2
TG_2	$1\ 0\ 1\ -1\ -2^0\ 0\ 0\ 0\ 0\ -2^0$	; keep TG_2
DG_3	$0\ 0\ 0\ 0\ 0\ -1\ 0\ 0\ 0\ 0$	; update DG_3
TG_2'	$1\ 0\ 1\ -1\ -2^0\ -1\ 0\ 0\ 0\ -2^0$	; *case* (c)

Thus, new TG_i's are : TG_1 ($-2^7 0 -2^7 0000000$) and TG_2 ($101-1-2^0-1000-2^0$). They are further CoMBined with DG_4.

$TG_1 * DG_4$:

TG_1	$-2^7\ 0\ -2^7\ 0\ 0\ 0\ 0\ 0\ 0\ 0$	
DG_4	$-4^3\ 0\ 0\ -4^3\ 0\ -4^3\ -4^3\ 0\ 0\ 0$	; *case* (f)
TG_1'	$-1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$	; $TG_1 * DG_4$ generates
TG_2'	$1\ 0\ -1\ -3^3\ 0\ -3^3\ -3^3\ 0\ 0\ 0$	; two terms TG_1' and TG_2'

$TG_2 * DG_4$:

TG_2	1 0 1 -1 -2^0 -1 0 0 0 -2^0	
DG_4	-4^3 0 0 -4^3 0 -4^3 -4^3 0 0 0	; *case* (b)
TG_2	1 0 1 -1 -2^0 -1 0 0 0 -2^0	; keep TG_2
DG_4	0 0 0 -3^3 0 -3^3 -3^3 0 0 0	; *case* (d)
TG_3'	1 0 1 -1 -2^0 -1 0 0 0 -2^0	; the result

Three TG_i's are produced: TG_1 (-1000000000), TG_2 (10-1-$3^3$0-3^3-$3^3$000), and TG_3 (101-1-2^0-1000-2^0).

To clarify these steps, we provide a step by step computation using Boolean notation. The four DG_i's are equivalent to $\overline{x_9x_7x_6}$, $\overline{x_9x_7x_5x_0}$, $\overline{x_9x_7x_4}$, and $\overline{x_9x_6x_4x_3}$. We have assumed that a DG_i is a function of Boolean variables 'x_0' through 'x_9'. The CMB operator determines

$(\overline{x_9x_7x_6})\ (\overline{x_9x_7x_5x_0})\ (\overline{x_9x_7x_4})\ (\overline{x_9x_6x_4x_3})$

$= (\overline{x_9x_7} + x_9x_7\ \overline{x_6}\ \overline{x_5x_0})\ (\overline{x_9x_7x_4})\ (\overline{x_9x_6x_4x_3})$

$= (\overline{x_9x_7}\ \overline{x_9x_7x_4} + x_9x_7\ \overline{x_6}\ \overline{x_5x_0}\ \overline{x_9x_7x_4})\ (\overline{x_9x_6x_4x_3})$

$= (\overline{x_9x_7} + (x_9x_7\ \overline{x_6}\ \overline{x_5x_0})\ (\overline{x_4}))\ (\overline{x_9x_6x_4x_3})$

$= (\overline{x_9x_7} + x_9x_7\ \overline{x_6}\ \overline{x_4}\ \overline{x_5x_0})\ (\overline{x_9x_6x_4x_3})$

$= (\overline{x_9x_7}\ \overline{x_9x_6x_4x_3} + x_9x_7\ \overline{x_6}\ \overline{x_4}\ \overline{x_5x_0}\ \overline{x_9x_6x_4x_3})$

$= \overline{x_9} + x_9\ \overline{x_7}\ \overline{x_6x_4x_3} + x_9x_7\ \overline{x_6}\ \overline{x_4}\ \overline{x_5x_0}$.

The above example is solved using following Boolean identities [53] :

$(\bar{x} + \bar{y})\ (\bar{x} + \bar{z}) = \bar{x} + \bar{y}\ \bar{z}$; $\bar{x}\ .\ \overline{x\bar{y}} = \bar{x}$; $\overline{x\bar{y}} = (\bar{x} + x\bar{y})$

GEN ($\oplus$) operator: Consider the path identifier F_j $(a_{l-1} \cdots a_1 a_0)$, the terms generated from **CASE 1** and **CASE 2** of the CMB operator as IG_η $(e_{l-1} \cdots e_1 e_0)$, and TG_k $(f_{l-1} \cdots f_1 f_0)$ where $a_i \in \{0, 1\}$, $e_i \in \{-\beta^I, 0\}$, and $f_i \in \{-\beta^I, 0, 1\}$. The GEN ($\oplus$) operator, then, obtains

$F^j = F_j \oplus IG_\eta \oplus TG_k$

As an example, assume F_j, IG_η, and TG_k. F^j is computed as

F_j	0	0	1	1	0	0	0	0	0	0
IG_η	-2^8	-2^8	0	0	0	0	-1	0	0	0
TG_k	0	0	0	0	1	-2^2	0	-2^2	1	0
F^j	-2^8	-2^8	1	1	1	-2^2	-1	-2^2	1	0

.

For $b_i \in F^j$, obtain $b_i = a_i \oplus e_i \oplus f_i$. The bitwise operation $\oplus$ is shown below :

$0 \oplus 0 \oplus 0 = 0$

$1 \oplus 0 \oplus 0 = 1$

$0 \oplus -\beta^I \oplus 0 = -\beta^I$

$0 \oplus 0 \oplus 1 = 1$

$0 \oplus 0 \oplus -\beta^I = -\beta^I$

In the above, we have considered only those five (out of 12) combinations which are feasible.

Theorem 3.1. The combinations (0, 1, 0) and $(-\beta^I, 0, 0)$ are not possible.

Proof: Using the definition of IG_η, it is easy to show that the occurrence of (0, 1, 0) is not possible. Moreover, F_j represents a path identifier defined over {0, 1}. Thus, a '$-\beta^I$' entry in F_j does not appear with F_j. □

Theorem 3.2. More than one occurrence of "1" or $-\beta^I$ in position i for the terms (F_j, IG_η, TG_k) is not feasible.

Proof: A multiple occurrence of "1" in position i confirms the presence of an uncomplemented variable with (F_j, TG_k) combination. The terms IG_η and TG_k represent disjoint expression and are generated for a path identifier F_j. Thus, the possibility of having two 1's in position i does not arise. A similar argument follows for multiple $-\beta^I$ or $(1, -\beta^I)$ combinations. □

3.4 CAREL: Algorithm and Analysis

3.4.1 Algorithm

The steps of the proposed algorithm are shown below:

```
CAREL:

  begin
    Sort F_i in ascending cardinality ;     /* refer to [75] for its advantages */
    F^1 = F_1 ;
    for (all minimal paths F_i) begin       /* i = 2, ··· */
      COM (F_i,F_j) ;                       /* j = 1, ··· , i-1; we get E_i's */
      RED (E_i's) ;                         /* return irredundant IG's and DG's */
      CMB (IG_i, DG_i) ;                    /* produce IG_η and TG_i */
      GEN (F_i, IG_η, TG_i) ;               /* get F^i's */
    end;
    Compute the reliability (unreliability) value ;

  end.
```

Consider COM1 and RED1 (COM2 and RED2) operations while using CAREL with Proposition P_I (P_II). Thus, CAREL applies equally for P_I and P_II. To compute the reliability (unreliability) parameter, we have developed a function called 're_num' which accepts the output from GEN (F_i, IG_η, TG_i). For given values of component reliability, re_num produces a numerical value for the reliability. One may use any other software package, such as 'vaxima' [85], to evaluate the reliability or unreliability expression.

Lemma 3.1. [33] Given cardinality ordered minimal (s,t)-paths of a network $G = (V, E)$ for 2CF measure, the e.m.d. term for any $(| V | - 1)$ sized minimal (s,t)-path P_i, is obtained directly by intersecting the complements of the remaining $\{| E | - (| V | - 1)\}$ links of G with P_i. □

This observation (first made in [60]) further reduces the computation time for algorithms based on SDP concepts. Lemma 3.1 is applicable only for the 2CF measure using minimal paths as the input. For 2CF with minimal paths, CAREL uses its four operators only for paths smaller than $| V | - 1$.

3.4.2 Analysis

This section describes an implementation of CAREL for proposition P_II. It is based on bit operations. The implementation of CAREL P_I is similar and will not be discussed. Both CAREL P_I and P_II are written in C. In what follows, we consider four macros which define bit operations. The cost for each macro call is also given. We also discuss the complexity issues involved in each of four operators described in Section 3.4.1.

a. SetUnion (s1,s2,s) ; /* s = s1 $\cup$ s2 */

cost : $\lceil l/16 \rceil$ *assignment* operations .

b. SetDif (s1,s2,s) ; /* s = s1 − s2 */

cost : $\lceil l/16 \rceil$ *xor* operations .

c. SetCompare (s1,s2) ; /* return *true* if s1 == s2 */

cost : $\leq \lceil l/16 \rceil$ *if* statements .

d. SubSet (s1,s2) ; /* return *true* if s1 $\subseteq$ s2 */

cost : 1 SetUnion ( ) + 1 SetCompare ( ) calls .

In a through d, l represents the number of components of the network, and a word w is of 16 bits. These four bit operations are used to implement CAREL operators.

COM (\ ) Operator:

A procedure **COM** (F_i, F_j) is implemented as follows :

for (j=1 to i−1) **begin**

SetUnion (F_i, F_j, s); /* s is temporary result */

SetDif (s, F_i, E_i'); /* E_i' is the result, append it to E_i' list */

end;

For a network with m minimal paths (cuts), the computation of COM operator is of the order of $O(m^2)$ since the operator is used $1 + 2 + \cdots + (m-1) = \frac{m(m-1)}{2}$ times.

RED (/) Operator:

The COM (\ ) operator produces E_k'. The E_k' and bit operations obtain non-redundant MCE's. The implementation of the RED operator is shown below:

```
for (all E_i's) begin      /* i = 1, ··· . Each E_i is in IG or DG group */
  if (SubSet(E_i', E_k') ) begin   /* E_k' is reducible by E_i' */
    dispose E_k'; break;
  end;
  else if (SubSet(E_k', E_i') )   /* E_i' is reducible by E_k' */
    dispose E_i';
end;
if (E_k' was not disposed)
  Put E_k' in IG or DG group;
```

Notice that the nonredundant E_i''s are in bit form (having 0's and 1's only). The MCE E_i's are generated from E_i''s by replacing '1' with $-\alpha^I$ in the positions of '1' where $\alpha \in \beta^I$ (refer to Section 3.3.2). The number of **RED** function calls is m times in a network with m minimal paths (cuts). The number of loopings inside the **RED** function depends on the network type and also on the path identifier F_i. In the worst case, **RED** for F_i needs i loopings, for $i = 1, 2, \cdots, m-1$, and hence the computation of the RED operator is of the order $O(m^2)$.

CMB (*) Operator:

The CMB (*) operator is the most time consuming operator of all the four operators we have used in our method. The implementation of CMB for the independent group is straightforward and is shown first.

```
for (all IG_i) begin        /* i = 1, ··· η−1 */
    for (j = 0 to l−1) begin
        if(IG_i[j] ≠ 0)
            IG_{i+1}[j] = IG_i[j];
    end;
end;
```

The implementation of CMB for the dependent group is expensive; each DG_i contains ($-\beta^I$, 0, 1). Here, we do not utilize bit operations for the CMB operator. To update the contents of the CMB operands, DG_i and TG_j, as well as to generate TG_j', we trace the contents of the operands element by element. The computation time is highly data dependent. But, in any *case* (refer to *cases* (a) through (f) in the procedure considered in Section 3.3.2), the order of computation time of the **DG** function is $O(l^2)$, where l is the number of components. In our program, **Step 1** of the algorithm falls into *case* (f). The maximum number of DG_i's is k for generating e.m.d. term(s) for F_k. Let g_i be the number of TG_i's created for a DG_i. Thus, the **DG** function is used $\sum_{i=1}^{k} g_i$ times. On the average, let us assume that g_i is equal to the number of e.m.d. terms generated for F_k. Then, the complexity of the CMB operator is $O(k \cdot | F^k |)$ of **DG** calls.

GEN ($\oplus$) Operator:

This operator is processed by sequentially tracing the contents of F_i and IG_η for generating F^i term(s). The procedure **GEN** (F_i, IG_η, TG_k) implements the $\oplus$ operator:

```
for (all TG_k's) begin        /* k = 1, ··· */
  for (j = 0 to l−1) begin
    if(F_i[j] ≠ 0)
      F^i[j] = F_i[j];
    else if(IG_η[j] ≠ 0)
      F^i[j] = IG_η[j];
    else
      F^i[j] = TG_k[j];
  end;
end;
```

For a path identifier F_i, we get disjoint term(s) F^i. The time complexity involved in **GEN** for a network depends on the number of the generated e.m.d. terms F^j's. If the maximum number of e.m.d. terms is M, the worst case complexity is of the order $O(mM)$.

3.5 Illustrating Examples

Example 3.3. Consider Figure 3.1 with its minimal (s,t)-paths x_1x_0, x_3x_2, $x_4x_3x_0$, and $x_4x_2x_1$. The minimal paths are encoded as path identifiers F_i's ($1 \le i \le 4$) and are sorted in their ascending cardinality as

F_1 (0000000000000011) F_3 (0000000000011001)
F_2 (0000000000001100) F_4 (0000000000010110)

Following the algorithm given in Section 3.4.1, $F^1 = F_1$. To generate e.m.d. term(s)

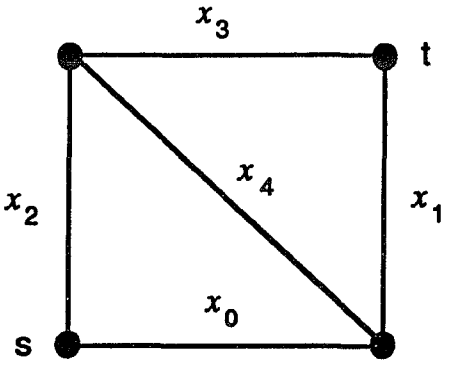


Figure 3.1. A Bridge Network

for F_3, we start with **COM** (F_3,F_1) and **COM** (F_3,F_2) which give E_1'(0000000000000010) and E_2' (0000000000000100), respectively. **RED** (E_i) retains both E_i's and puts them into IG with an empty DG group. CMB operation obtains IG_η as 0000000000000-1-10. Finally, **GEN** $(F_3, IG_\eta, \varnothing)$ gives F^3 as (00000000000011-1-11), which can be interpreted in an intermediate form as $x_4x_3\overline{x_2}\,\overline{x_1}x_0$. This resultant expression has one to one correspondent with the probability expression. F^2 and F^4 can be obtained in a similar manner as in the F^3. The various e.m.d. terms are

$$F(disjoint) = x_1x_0 + x_3x_2\,\overline{x_1x_0} + x_4x_3\overline{x_2}\,\overline{x_1}x_0 + x_4\overline{x_3}\,x_2x_1\overline{x_0}.$$

Example 3.4. Consider a 3-cube shown in Figure 3.2 and its six diameter distance minimal (s,t)-paths, $x_{10}x_1x_0$, $x_9x_5x_0$, $x_{10}x_3x_2$, $x_{11}x_6x_3$, $x_8x_7x_6$, and $x_8x_5x_4$. Their path identifiers are

F_1 (0000010000000011) F_2 (0000001000100001) F_3 (0000010000001100)
F_4 (0000100001001000) F_5 (0000000111000000) F_6 (0000000100110000).

The e.m.d. term(s) for F_4 is (are) generated using the steps mentioned in the algorithm. The details are shown below :

For $1 \le i \le 3$, **COM**(F_4, F_i) obtains E_1' (0000010000000011), E_2' (0000001000100001), and E_3' (0000010000000100). The RED operation fails to remove any E_i's because there is no redundant terms. The RED classifies these terms into DG with an empty IG set. Thus, the CMB operator obtains IG_η as 000000000000000. Using **Step 1**, the elements in DG are used to generate TG_1 and TG_2 as follows:

DG_1	0 0 0 0 0 0 -3^0 0 0 0 0 0 0 0 0 -3^0 -3^0
DG_2	0 0 0 0 0 0 0 -3^0 0 0 0 -3^0 0 0 0 0 -3^0
TG_1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
TG_2	0 0 0 0 0 0 -2^1 -2^5 0 0 0 -2^5 0 0 0 -2^1 1 .

Using **Step 2** and **DG**, we have **DG**(TG_1, DG_3, TG') and **DG**(TG_2, DG_3, TG').

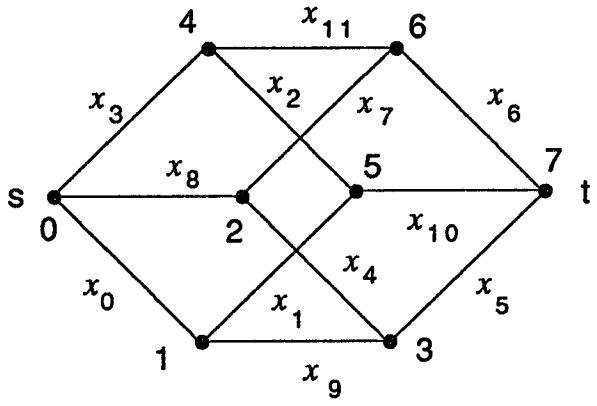


Figure 3.2. A 3-cube

The first **DG** is computed as follows:

$$
\begin{array}{ll}
TG_1 & 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1 \\
DG_3 & \underline{0\ 0\ 0\ 0\ 0\ -2^2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -2^2\ 0\ 0} \\
TG_1' & 0\ 0\ 0\ 0\ 0\ -2^2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -2^2\ 0\ -1\ .
\end{array}
$$

The second **DG** obtains:

$$
\begin{array}{ll}
TG_2 & 0\ 0\ 0\ 0\ 0\ -2^1\ -2^5\ 0\ 0\ 0\ -2^5\ 0\ 0\ 0\ -2^1\ 1 \\
DG_3 & \underline{0\ 0\ 0\ 0\ 0\ -2^2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -2^2\ 0\ 0} \\
TG_2' & 0\ 0\ 0\ 0\ 0\ -1\ -2^5\ 0\ 0\ 0\ -2^5\ 0\ 0\ 0\ 0\ 1 \\
TG_3' & 0\ 0\ 0\ 0\ 0\ 1\ -2^5\ 0\ 0\ 0\ -2^5\ 0\ 0\ -1\ -1\ 1\ .
\end{array}
$$

Finally, **GEN**(F_4, IG_η, TG_j's) generates F^4 as (00001-$2^2$0001001-$2^2$0-1), (00001-1-$2^5$001-$2^5$01001), (000011-$2^5$001-$2^5$01-1-11), which can be interpreted as $x_{11}x_6x_3\ (\overline{x_{10}x_2}\ \overline{x_0} + \overline{x_{10}}\ \overline{x_9x_5}x_0 + x_{10}\overline{x_9x_5}\ \overline{x_2}\ \overline{x_1}x_0)$. The remaining e.m.d. terms for this example are obtained similarly, and we generate the disjoint expression as

$$
\begin{aligned}
F(\mathit{disjoint}) = {} & x_{10}x_1x_0 + x_9x_5x_0\ (\overline{x_{10}x_1}) + x_{10}x_3x_2\ (\overline{x_0} + \overline{x_9x_5}\ \overline{x_1}x_0) \\
& + x_{11}x_6x_3(\overline{x_{10}x_2}\ \overline{x_0} + \overline{x_{10}}\ \overline{x_9x_5}x_0 + x_{10}\overline{x_9x_5}\ \overline{x_2}\ \overline{x_1}x_0) \\
& + x_8x_7x_6(\overline{x_3}\ \overline{x_0} + \overline{x_{11}}x_3\overline{x_{10}x_2}\ \overline{x_0} + \overline{x_{10}}\ \overline{x_9x_5}\ \overline{x_{11}x_3}x_0 \\
& + x_{10}\overline{x_9x_5}\ \overline{x_3}\ \overline{x_1}x_0 + \overline{x_{11}}x_{10}\overline{x_9x_5}x_3\overline{x_2}\ \overline{x_1}x_0) \\
& + x_8x_5x_4(\overline{x_7x_6}\ \overline{x_3}\ \overline{x_0} + \overline{x_6}x_3\overline{x_{10}x_2}\ \overline{x_0} + \overline{x_{11}}\ \overline{x_7x_6}x_3\overline{x_{10}x_2}\ \overline{x_0} \\
& + \overline{x_{10}}\ \overline{x_9}\ \overline{x_6}x_0 + \overline{x_{10}}\ \overline{x_9}\ \overline{x_7x_6}\overline{x_{11}x_3}x_0 + x_{10}\overline{x_9}\ \overline{x_7x_6}\ \overline{x_3}\ \overline{x_1}x_0 \\
& + x_{10}\overline{x_9}\ \overline{x_6}x_3\overline{x_2}\ \overline{x_1}x_0 + \overline{x_{11}}x_{10}\overline{x_9}\ \overline{x_7x_6}x_3\overline{x_2}\ \overline{x_1}x_0).
\end{aligned}
$$

Example 3.5. In this example, we show how to compute the probabilistic k-subcube for the hypercube system, where the system is said to be functional as long as a k-subcube is operational. Consider a 3-cube shown in Figure 3.2. The 3-cube contains six 2-cubes: {0, 1, 2, 3}, {0, 1, 4, 5}, {0, 2, 4, 6}, {1, 3, 5, 7}, {2, 3, 6, 7}, and {4, 5, 6, 7}. We are interested in obtaining the probability that there exists at least one 2-cube in the 3-cube given that nodes fail with probability q. Let

Boolean variable x_i denote a node i. Using this information as the input to CAREL and considering $q = 0.1$, we obtain the mutually disjoint terms for the probabilistic 2-subcube functionality measure as

$$F(disjoint) = x_3x_2x_1x_0 + x_5x_4x_1x_0(\overline{x_3}x_2) + x_6x_4x_2x_0(\overline{x_1} + \overline{x_5}\,\overline{x_3}x_1)$$
$$+ x_7x_5x_3x_1(\overline{x_0} + \overline{x_4}\,\overline{x_2}x_0) + x_7x_6x_3x_2(\overline{x_5x_1}\,\overline{x_0} + \overline{x_4}\,\overline{x_1}x_0)$$
$$+ x_7x_6x_5x_4(\overline{x_3}\,\overline{x_0} + x_3\,\overline{x_2}\,\overline{x_1}\,\overline{x_0} + \overline{x_2}\,\overline{x_1}x_0).$$

Substituting probability of x ($\overline{x}$) with 0.9 (0.1), we find the measure as 0.955216.

3.6 Shellable Systems

This section discusses a class of structures whose reliability is computed in time polynomial in the order of the number of their minimal paths. To begin, we present some definitions that will be used throughout our discussion. Let OC (FC) represent the set of operative (failed) components; AC = OC $\cup$ FC gives the entire system components. Note, OC $\cap$ FC = $\varnothing$.

Definition. A system is a coherent binary system [14] if:

1. Each component of the system can be in either of two states: operative or failed.
2. The system takes on the two states (operative or failed) as a function of its component states.
3. When the system has failed, no component failure restores the system to operative state.
4. When the system is operative, no component repair causes the system to fail.
5. Failure (operation) of all system components leads to system failure (operation).

Let Ω denote the set of all the paths for the problem under consideration. A coherent binary system (AC, Ω) is a rank d matroid [20] if all elements of Ω have cardinality d and for each $P_i, P_j \in \Omega$ and each $\alpha \in P_i - P_j$ there exists a

$\beta \in P_j - P_i$ such that $P_j \cup \{\alpha\} - \{\beta\} \in \Omega$. For a connected graph $G = (V, E)$, for AC (Ω) as the set of links (spanning trees) in G, a coherent binary system (AC, Ω) is a rank $|V|-1$ matroid. Note, the network reliability problem computes the probability that at least one spanning tree exists in the system. A simple product is defined as

$$[\mathrm{OC},\mathrm{FC}] = \bigcap_{j\in \mathrm{OC}} x_j \bigcap_{j\in \mathrm{FC}} \overline{x_j}.$$

Assuming that components fail independently, the probability of the simple product is given as

$$\mathbf{Pr}([\mathrm{OC},\mathrm{FC}]) = \prod_{j\in \mathrm{OC}} p_j \prod_{j\in \mathrm{FC}} (1 - p_j),$$

where p_j is the reliability of component j.

Consider the disjoint product concept discussed in Subsection (3.1), in particular, Equation (3.3). The reliability contribution of the ith term of the equation, i.e., $\mathbf{Pr}(F_i\overline{F_1} \cdots \overline{F_{i-1}})$, is given in Equation (3.4) as

$$\mathbf{Pr}(F_i) \cdot \mathbf{Pr}(E_1E_2 \cdots E_{i-1}). \tag{3.9}$$

The efficiency of the Boolean technique depends on the efficiency of computing Expression (3.9). Assuming the independence of component failures,

$$\mathbf{Pr}(F_i) = \mathbf{Pr}([F_i,\varnothing]) = \prod_{j\in F_i} p_j.$$

Unfortunately, computing $\mathbf{Pr}(E_1E_2 \cdots E_{i-1})$ is not as straightforward. In general, these E_i's are not mutually independent and hence,

$$\mathbf{Pr}(E_1E_2 \cdots E_{i-1}) \neq \prod_{j=1}^{i-1} \mathbf{Pr}(E_j). \tag{3.10}$$

However, the equality for Equation (3.10) is guaranteed if (E_j, E_k) pair is mutually independent for $j, k = 1, 2, \cdots, i-1$.

Definition. A *shelling* [20] is a coherent binary system in which there is an ordering of minimal paths $P_1, P_2, \cdots, P_m$ so that for each P_i, setting $\mathbf{Pr}(E_j) = \mathbf{Pr}(\overline{F_j} \mid F_i)$, for each $1 \leq j \leq i-1$, we obtain equality for Equation (3.10).

Definition. A coherent binary system (AC, Ω) is shellable if there exists a shelling ordering for Ω.

The problem of testing if a given ordering of Ω is a shelling can be done in polynomial time in the size of Ω. However, obtaining the shelling ordering for a given set Ω is still an open problem [8]. Ball and Provan [8] show that a matroid is a shellable system. The probabilistic NCF problem for the hypercube corresponds to the coherent binary system (AC, Ω) in which AC (Ω) is the set of links (spanning trees). For hypercube, (AC, Ω) is a rank $N-1$ matroid.

Example 3.6. Consider the network shown in Figure 3.3 with its sixteen minimal all-paths given as (0 1 2), (0 1 4), (1 3 5), (2 3 4), (0 4 5), (0 1 3), (0 2 3), (0 2 4), (0 2 5), (1 2 3), (1 2 5), (1 3 4), (1 4 5), (0 3 5), (3 4 5), and (2 4 5). For AC = {0, 1, 2, 3, 4, 5}, and Ω as the set of these sixteen spanning trees, (AC, Ω) represents a rank 3 matroid and hence is shellable. However, the paths are not in a shelling ordering; there are eighteen e.m.d. terms generated for the input.

Example 3.7. In this example, we preprocess the sixteen minimal all-paths into the following ordering: (1 3 5), (0 1 2), (0 1 4), (0 4 5), (2 4 5), (1 2 5), (0 3 5), (3 4 5), (2 3 4), (1 2 3), (0 1 3), (1 3 4), (1 4 5), (0 2 3), (0 2 4), and (0 2 5). For this input, we obtain exactly sixteen e.m.d. terms which indicate that the minimal paths are in shelling ordering. A lexicographic ordering of the minimal paths is another shelling ordering.

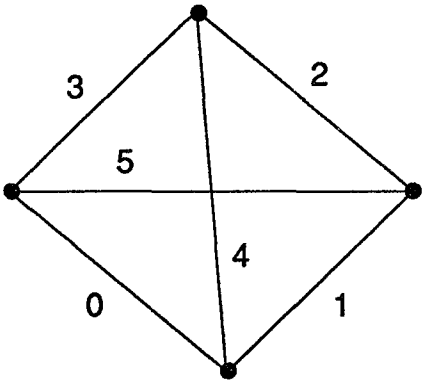


Figure 3.3. An Example Network

Lemma 3.2. Given a shelling ordering of minimal paths in Ω, CAREL computes the reliability of the coherent binary system (AC, Ω) in time polynomial in the order of $| \Omega |$.

Proof: A matroid is shellable [8]. By definition of a shellable system, a shelling ordering of the minimal paths in Ω will obtain mutually independent E_i's, and, thus, all of these E_i's belong to the IG group (refer to the CMB operator discussed in Section 3.3.2). Obviously, minimal paths preprocessed in shelling ordering can be transformed into mutually disjoint terms by CAREL in time polynomial in the order of the number of minimal paths. □

To obtain a shelling ordering for the minimal all-paths of a 3-cube, we lexicographically order the 384 spanning trees of the C_3. Then, we use CAREL to obtain the mutually disjoint terms for the ordered minimal paths. CAREL generates exactly 384 e.m.d. terms for the ordered input set. Thus, the lexicographic ordering of the minimal paths of C_3 is a shelling ordering. We do not generate the same experiment on the 4- or higher dimension cubes since the number of minimal all-paths for these cubes are in the order larger than 10^{10} (refer to Equation (2.1)). Since a hypercube is regularly structured, we conjecture that lexicographic ordering of the minimal all-paths of a cube with any dimension will give the shelling ordering, and, hence, CAREL solves the probabilistic NCF measure for the hypercube in time polynomial in the number of its spanning trees.

3.7 Conclusion

We have presented an efficient algorithm called CAREL which computes the terminal reliability or unreliability of moderate to large sized networks with modest memory and time requirements [74,75]. The algorithm has been implemented in C

and was run on an Encore MULTIMAX 320 system. The performance of a program is usually based on the algorithm, the data structure and the language used, the computer on which the program is run, and, last but not least, the coding of the program [4]. Since different programmers produce different codings for an algorithm, the human factor (in want of sufficient data) is inappropriate while comparing various techniques. A better implementation or a faster machine would increase the performance of a program but only to a constant factor [4]. CAREL is faster than other existing Boolean algorithms [74], and this is obvious from the CPU time requirement for solving the terminal reliability / unreliability of various computer networks (see Reference [74]). An earlier version of CAREL has successfully been applied to solve reliability problems in one type of redundant path multistage interconnection network [64]. All exact reliability evaluation problems are known to be computationally intractable or NP-hard, which makes comparing the techniques from the aspect of complexity difficult [8,11,20,62].

CAREL is shown to run in time polynomial in the order of the number of minimal paths for shellable systems, such as the probabilistic model with NCF measure in the hypercube. In a hypercube system, the number of spanning trees increases superexponentially in the order of the cube dimensions (see Equation (2.1)). Thus, CAREL is not suggested for use in evaluating the probabilistic NCF measure for cubes of large dimensions. For general networks, the total number of basic (s,t)-paths are much less than that of minimal (s,t)-paths [40], and, therefore, reliability computation from basic paths is much more efficient than that from minimal paths. However, in the hypercube, there are exactly $n!$ of either diameter-distance basic (s,t)-paths or minimal (s,t)-paths. It is obvious that the basic path concept does not help reduce the time complexity of the 2CF reliability problem for the hypercube. In Chapter 4, we present lower bounds values for the probabilistic

model with NCF and 2CF measures and also show that the proposed methods are polynomial in the order of cube dimensions.

Chapter 4

Reliability Bounds for Hypercube Networks

As the size of the hypercube network increases, exact reliability modeling for the network becomes more complex. The general problem of computing probabilistic measures is NP-hard [20,29,62], and, thus, analyzing large hypercubes will require an unreasonable amount of computation time. This leads to the development of methods for approximating the reliabilities using lower and upper bounds, between which the exact measure is guaranteed to exist. Methods for generating the bounds are of particular interest as they are computed efficiently. The lower bound on the system reliability is quite appealing because it provides the probability that C_n will be at least this reliable. If the lower bound indicates that a system will be operational over the time interval of interests, then it is unnecessary to obtain an exact reliability measure. The upper bound, on the other hand, provides an optimistic assurance of the system functionality. Any approximation reliability results which give more favorable reliability figures compared to those of the upper bound are impossible and are discarded. Thus, the upper bounds on reliability are used to check the accuracy of the approximation results. Moreover, when the lower and upper bounds on reliability are tight, the results give a better indication of the exact reliability (compared to knowing only the lower bound reliability figure). In this chapter, we present efficient techniques to obtain the lower and/or upper bounds on probabilistic models with NCF, 2CF, and TBF measures.

To obtain bounds on the probabilistic network-connected functionality (NCF) measure for a hypercube, we first consider the reliability polynomial concept [20]. The reliability polynomial is a graph invariant which is of interest in cases where graphs are used as models of systems such as computer networks. A method based on this concept runs in time polynomial in the number of CE's and is exponential in the dimension of the n-cube. Furthermore, as will be shown later, such bounds are not tight. Yang *et al.* [88] described a lower bound on the NCF measure by obtaining the number of spanning trees in C_n and then weighted each spanning tree by considering its (2^n-1) links as operational and the rest of the link set as failed. Their bound is acceptable if the link reliability, p, is very small [88]. Bulka and Dugan [15] computed a lower bound on the NCF measure for an n-cube starting from a lower bound on the measure for an $(n-1)$-cube, which in turn is obtained from that of an $(n-2)$-cube, and so on, until the cube is small enough to be a *base* cube whose NCF parameter is evaluated directly. Their method uses a C_2 as the base cube for which the exact NCF reliability measure is given as

$$NCF(2, p) = 4p^3(1-p) + p^4.$$

Reference [15] also provides an efficient method which runs in time exponential in the order of n. In this chapter, we first review the recursive technique by Bulka and Dugan [15] (henceforth called the BD approach), then modify it to speed up its evaluation. It is shown that the modified technique is computable in time polynomial in the order of n. We next establish a recursive method that further utilizes the structural properties of the hypercube to produce a tighter lower bound. A method in [26] can be used to obtain the upper bound on NCF reliability.

A lower bound on the probabilistic 2-connected functionality (2CF) measure is discussed in the literature. For 2CF, let the specified nodes s and t be at distance n in a w-cube, where $n \leq w$. Without loss of generality, consider t as an antipodal

node to s in an n-subcube. Thus s and t represent farthest nodes and the shortest path between them is of length n. We will lower bound 2CF measure by examining only fixed-distance paths of length n between s and t. Previous attempts to bound 2CF metric have also assumed fixed-distance paths. Assuming CE failures only, Latifi [47] derived a lower bound on 2CF measure by considering w disjoint paths in the C_w comprising n paths of length n and $w-n$ paths of length $n+2$. The result does not provide a tight bound with large n. Ahmed and Trahan [3] considered all $n!$ shortest paths, but the method requires excessive time. In this chapter, we introduce a method that efficiently evaluates $O(n^2)$ shortest paths, providing a significantly improved bound than that obtained in reference [47], while still computable in reasonable time. An alternative approach, based on a two-cube (TC) model of an n-cube, is also introduced. It is shown that the TC technique performs well with smaller $r(p)$, where $r(p)$ is the probability of working of a node (link) in a hypercube. Our approach for tighter bound computation derives the results from these two methods.

This chapter also describes a method for task-based functionality (TBF) reliability evaluation in hypercube with failing nodes. Najjar and Gaudiot [54,55] presented a model for a hypercube with up to 50% system degradation. Their method starts with PE disconnection probability which in turn is used to approximate the reliability metric. Kim *et al.* [42] offered an approach based on a decomposition technique where a large hypercube is recursively divided into smaller cubes until a cube can be evaluated for its exact reliability. Both Najjar-Gaudiot's and Kim *et al.*'s methods run in exponential time in the order of the cube dimension. In this chapter, we present an efficient technique to obtain an upper bound on TBF for an n-cube and compare our bounds with those obtained by the methods presented in [42,54].

The subcube functionality (SF) metric is important because many hypercube algorithms are executed on various sizes of subcubes by setting parameters appropriately [51,59]. Bounds on probabilistic SF measure are also described in the literature. Abraham and Padmanabhan [2] studied the probability of the existence of disjoint functional subcubes of different sizes when all faulty PE's and/or CE's are contained in a subcube. This model, in essence, is a probabilistic model with SF criteria. Das and Kim [21] discuss a generalized approach for the model, but this dissertation does not address this measure.

4.1 Reliability Polynomial Concept

Let N_j be the number of paths having exactly j operating links in the network $G = (V, E)$. The reliability polynomial, $Rel(G, p)$, of a network G with L links is expressed as [20]

$$Rel(G, p) = \sum_{j=0}^{L} N_j p^j q^{L-j}, \tag{4.1}$$

where p denotes the link reliability and $p+q \equiv 1$. Alternatively, the network cuts are used to express the reliability polynomial. Let Q_j be the number of cuts with exactly j links. Using cuts, we have

$$Rel(G, p) = 1 - \sum_{j=0}^{L} Q_j q^j p^{L-j}.$$

We define f-set $\subseteq E$ to denote a set of links whose complement is a path. Using the f-set concept, the reliability polynomial is given as

$$Rel(G, p) = \sum_{j=0}^{L} F_j q^j p^{L-j},$$

where F_j represents the number of f-sets having j links. The polynomial representation translates the reliability concept into a problem of enumerating the number of

paths, cuts, or f-sets of a network. In general, this problem is of exponential complexity in the size of the network. In what follows, we consider an application of the polynomial representation for the bounds on the probabilistic NCF measure with CE failures only.

Let $S \subseteq E$ be a set of CE's in the network G. If $E-S$ is a path, then S is an f-set; otherwise, S represents a cut. Thus, $F_i + Q_i = \binom{L}{i}$ [20]. For the number of minimal cuts in G as Q_c, $F_c = \binom{L}{c} - Q_c$. The number Q_c is computed in polynomial time [20] and, hence, F_c is also obtainable in polynomial time. Since c is the cardinality of the minimal cut, the network G would not have any cut of size less than c, i.e., $Q_i = 0$ for all $i < c$. Thus, $F_i = \binom{L}{i}$ for $i < c$. Let l be the cardinality of the minimal path for G, and let $d = L - l$. In other words, d is the maximum number of CE's that can be removed from G without disconnecting G, and the set of the remaining CE's is the minimal path in G. Thus, if we remove more than d CE's from E, there is no spanning tree in G, i.e., $F_i = 0$ for $i > d$. F_d is the number of spanning trees in G and is computable in polynomial time. The remaining F_i's that need to be computed are $F_{c+1}, F_{c+2}, \cdots, F_{d-2}, F_{d-1}$.

Each F_i is evaluated by some approximation technique that makes use of the *hereditary* structure of the f-set family [20]. A family of sets is said to be *hereditary* if and only if for every set S in the family, all subsets of S are also members of the family. For the f-set family, every subset of an f-set is itself an f-set (by the definition of the f-set), and, therefore, f-set family is *hereditary*. Lower and upper bound values for the unknown F_i's are computed using Sperner or Kruskal-Katona bounds. Reference [20] provides an extensive discussion on these bounds for general networks.

4.2 NCF Bounds - A Review

In this section we present a review on previous techniques to compute the bounds on NCF measure for the hypercube. First, Sperner and Kruskal-Katona's theorems are used with the polynomial representation to compute lower and upper bounds on the NCF measures. Then, Bulka-Dugan approach is presented.

4.2.1 Bounds Using Polynomial Representation

A. Sperner bounds are given as [20]

$$F_{i-1} \geq \frac{i}{L-i+1} F_i \tag{4.2}$$

Equation (4.2) can be used to compute bounds on the probabilistic NCF measures. For an n-cube, $N = 2^n$, $L = n \cdot 2^{n-1}$, and $d = L-N+1$. A lower bound on the NCF measure is computed starting from F_d, which is the number of spanning trees, $ST(n)$, of the n-cube. We recursively apply Equation (4.2) to get F_{d-1} from F_d, F_{d-2} from F_{d-1}, and so on, until F_{n+1}, the values of which are used to compute the lower bound of the NCF measure. Thus, an expression for the lower bound on NCF reliability of a hypercube is given as

$$NCF(n,p) \geq \sum_{j=0}^{n-1} \begin{bmatrix} L \\ j \end{bmatrix} p^{L-j} q^j + F_n p^{L-n} q^n + \sum_{j=n+1}^{d} F_d \frac{\begin{bmatrix} L \\ j \end{bmatrix}}{\begin{bmatrix} L \\ d \end{bmatrix}} p^{L-j} q^j. \tag{4.3}$$

Similarly, we derive an expression for the upper bound reliability on NCF using (4.2) and starting from $F_{i-1} = F_n$, which is given as $\begin{bmatrix} L \\ n \end{bmatrix} - 2^n$. F_n is used to compute F_{n+1} which, in turn, is utilized to compute F_{n+2}, and so on, until F_{d-1} is obtained. The upper bound, then, is given as

$$NCF(n,p) \le \sum_{j=0}^{n-1} \binom{L}{j} p^{L-j} q^j + \sum_{j=n}^{d-1} F_n \frac{\binom{L}{j}}{\binom{L}{n}} p^{L-j} q^j + F_d p^{N-1} q^d. \qquad (4.4)$$

Example 4.1. Consider a 3-cube in which $L = 12$, $F_5 = 384$ and $F_3 = 212$. Using (4.2), the lower and upper bounds of F_4 are computed as 240 and 522, respectively. The exact value for F_4 of a 3-cube is 408. Using Equations (4.3), (4.4) and CE reliability $p = 0.9$, the lower and upper bounds on NCF for the 3-cube are computed as 0.9834310 and 0.9936331, respectively.

B. Kruskal-Katona bounds use the concept of f-vector[20] and hereditary family (refer to Section 4.1), and are given as [20]:

$$F_{k-1} \ge F_k^{k-1/k}, \; 0 < k \le d \; , \qquad (4.5)$$

where $(F_0, F_1, \cdots, F_d)$ represent the f-vector for a hereditary family. Note,

$$F_k^{k-1/k} = \binom{a_k}{k-1} + \binom{a_{k-1}}{k-2} + \cdots + \binom{a_1}{0}, \qquad (4.6)$$

where $(a_k, a_{k-1}, \cdots, a_1)$ is defined as the k-canonical form of F_k. This k-canonical form is computed as

$$F_k = \binom{a_k}{k} + \binom{a_{k-1}}{k-1} + \cdots + \binom{a_1}{1}.$$

For a given F_k, the k-canonical form is generated, and an $F_k^{k-1/k}$ is the lower bound for F_{k-1}.

The Kruskal-Katona lower and upper bounds on the probabilistic NCF measure for a C_n can be derived from (4.5) as

$$NCF(n,p) \ge \sum_{j=0}^{n-1} \binom{L}{j} p^{L-j} q^j + F_n p^{L-n} q^n + \sum_{j=n+1}^{d} F_d^{j/d} p^{L-j} q^j \qquad (4.7)$$

$$NCF(n,p) \le \sum_{j=0}^{n-1} \begin{bmatrix} L \\ j \end{bmatrix} p^{L-j} q^j + \sum_{j=n}^{d-1} F_n^{j/n} p^{L-j} q^j + F_d p^{N-1} q^d. \qquad (4.8)$$

Example 4.2. Consider a C_3 where $F_5 = 384$. We use Kruskal-Katona lower bound to obtain F_4. The 5-canonical form of F_5 is given as (a_5=10, a_4=9, a_3=4, a_2=2, a_1=1); $F_5^{4/5}$ is 303, and, hence, the lower bound $F_4 = 303$. Note, the exact value for F_4 is 408. Using Equation (4.7) and $p = 0.9$, we compute the lower bound on NCF for the C_3 as 0.9861430.

Similarly, given F_c, we are able to generate the upper bound of F_{c+1} since $F_{c+1} \le F_c^{c+1/c}$. Consider $F_3 = 212$ with its 3-canonical form as (a_3=11, a_2=10, a_1=1). $F_3^{4/3}$ is computed as $\begin{bmatrix} 11 \\ 4 \end{bmatrix} + \begin{bmatrix} 10 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 450$, which gives the upper bound value for F_4. Note, for $i \le j$ we define $\begin{bmatrix} i \\ j \end{bmatrix} = 0$. Using Equation (4.8) with $p = 0.9$, we obtain the upper bound on NCF for the 3-cube as 0.9925879.

In general, the Kruskal-Katona's bounds are tighter than the Sperner's bounds [20]. However, both approaches fail to give tight bounds on NCF for n-cubes with $n > 4$. Using CE reliability $p = 0.9$, the lower bounds on NCF computed by either method deteriorate rapidly to zero and the upper bounds move close to 1. Obviously, a better approach for evaluating tighter bounds on NCF is needed. Furthermore, the methods based on polynomial expression are polynomial only in the order of the number of CE's, and, thus, are exponential in the order of the cube dimension.

4.2.2 Bulka-Dugan Approach

Bulka and Dugan [15] presented a lower bound on the NCF measure of an n-cube recursively computed from a lower bound on the reliability of two component $(n-1)$-cubes. They used a C_2 as the base cube. In the following, a C_n is said to be decomposed into two *congruent* C_{n-1}'s such that each node on one C_{n-1} is linked to its congruent node on the other C_{n-1}. All 2^{n-1} links connecting two C_{n-1}'s are termed as *exterior* links, and the links within each C_{n-1} as *interior* links. The BD algorithm [15], given below, computes a lower bound on NCF measure of a C_n by computing the probability of three disjoint cases of working and failed congruent $(n-1)$-cubes and exterior links.

Case 1. Both $(n-1)$-cubes are operating and i exterior CE's operate, for $1 \le i \le 2^{n-1}-2$.

$$T1 = NCF(n-1, p)^2 \cdot \sum_{i=1}^{2^{n-1}-2} \binom{2^{n-1}}{i} p^i q^{2^{n-1}-i}$$

Case 2. At least one $(n-1)$-cube is operating, and one exterior CE is failed. The node at the end point of the failed link in the non-operating C_{n-1} is linked to at least one of its neighbors by an interior link.

$$T2 = [2 \cdot NCF(n-1, p)(1-q^{n-1}) - NCF(n-1, p)^2] \cdot 2^{n-1} p^{2^{n-1}-1} q$$

Case 3. All 2^{n-1} exterior CE's operate.

$$T3 = NCF(n-1, p^2+2pq) \cdot p^{2^{n-1}}$$

A lower bound on NCF reliability of an n-cube is obtained as

$$NCF(n, p) = T1 + T2 + T3. \tag{4.9}$$

Computing Case 1 takes exponential time in the dimension of the cube, and thus is useful only for small sized cubes. For large n, Bulka and Dugan introduced

another approach in which an n-cube is viewed as a 1-cube with two (2^{n-1})-supernodes connected by one (2^{n-1})-link. They next generalized the approach by considering the n-cube as a 2-cube whose four nodes are each $(n-2)$-cubes, or a 3-cube whose eight nodes are each $(n-3)$-cubes, and so on. In general, an n-cube is viewed as an $(n-k)$-cube whose 2^{n-k} nodes are each k-cubes that are connected by (2^k)-links. Therefore, if the original link reliability is p, the new link reliability is $1-(1-p)^{2^k}$, which is the probability that at least one of the 2^k links is operating. Using this approach, Bulka and Dugan [15] obtained another lower bound on the NCF measure of an n-cube as

$$NCF(n, p) = NCF(k, p)^{2^{n-k}} \cdot NCF(n-k, 1-(1-p)^{2^k}). \tag{4.10}$$

For large k and p ($k \geq 2$ and $p \geq 0.9$), the second term of the right hand side in Equation (4.10) is approximately equal to 1. Thus, a simplified version of Equation (4.10) is given as

$$NCF(n, p) = NCF(k, p)^{2^{n-k}}. \tag{4.11}$$

Equation (4.9) provides a tighter bound compared to those obtained by Equations (4.10) and (4.11) [15].

4.3 Polynomial-time Algorithm

4.3.1 Modified Bulka-Dugan Approach

In what follows, we suggest two improvements on Equation (4.9). First, the combinatorial identity $\sum_{i=0}^{2^{n-1}} \binom{2^{n-1}}{i} p^i q^{2^{n-1}-i} = 1$ is used to express $T1$ as

$$T1 = NCF(n-1, p)^2 \cdot (1 - q^{2^{n-1}} - p^{2^{n-1}} - 2^{n-1}p^{2^{n-1}-1}q\ ). \tag{4.12}$$

Equation (4.12) can be computed in time polynomial in the dimension of the cube,

and hence, we need not use the inferior lower bounds obtained by Equations (4.10) or (4.11). Second, since Case 3 requires computation of NCF with different link reliability, this causes overall $O(n^2)$ recursive calls to be evaluated to compute NCF. We compute $T3'$, a lower bound on $T3$, in such a way as to reduce the overall number of recursive calls to n.

Case 3′. At least one $(n-1)$-cube operates and all 2^{n-1} exterior links operate.

$$T3' = [NCF(n-1, p)^2 + 2 \cdot NCF(n-1, p) \cdot (1 - NCF(n-1, p))] \cdot p^{2^{n-1}}.$$

It is found that the resultant lower bound is still tight for cubes having moderate to large dimension sizes. The Bulka-Dugan approach and our modification to it, however, fail to give reasonably good results when the failure rate increases (over a period of concurrent use-time of the system). In what follows, we present a new algorithm which is not only computationally efficient but also provides a tighter bound on NCF measure as compared to existing techniques.

4.3.2 An Efficient Technique for NCF Bound

Let us consider a C_n as two $(n-1)$-cubes connected by 2^{n-1} CE's. We evaluate the lower bound on NCF reliability of a C_n from the known lower bound on NCF reliability of a C_{n-1}. The proposed algorithm divides the problem into three mutually disjoint cases. In addition, the events in each case are also mutually disjoint among themselves. Thus, the lower bound on NCF reliability can be calculated as the sum of the lower bounds on NCF reliability obtained from the following three cases.

CASE 1: Both $(n-1)$-cubes and at least one exterior link operate.

The graph model for CASE 1 is shown in Figure 4.1. Since both C_{n-1}'s operate, only one good exterior CE is needed to make the two subcubes combine into a

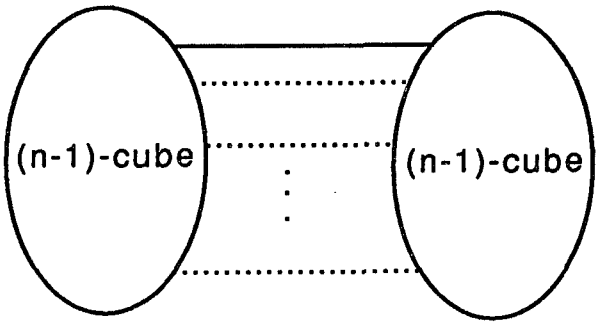


Figure 4.1 An n-cube with One Healthy Exterior Link

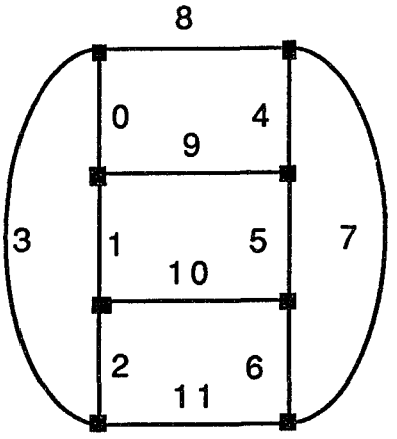


Figure 4.2 A 3-cube with All Its Labeled Components

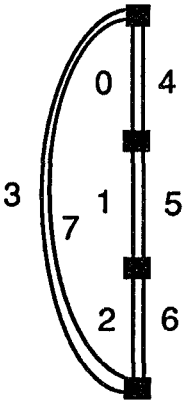


Figure 4.3 A 3-cube with All Healthy Exterior Links

connected C_n. The disjoint expression of CASE 1 is given as:

$$NCF(n-1, p)^2 \cdot (1 - q^{2^{n-1}}). \tag{4.13}$$

For CASE 1, a spanning tree is constructed from two spanning trees, one from each of the two C_{n-1}'s, connected by an exterior link. Each of the two $(n-1)$-cubes has $ST(n-1)$ spanning trees, and there are 2^{n-1} exterior links. Thus, CASE 1 includes $ST_{CASE\,1}(n) = 2^{n-1} \cdot ST(n-1) \cdot ST(n-1)$ spanning trees, where $ST_x(n)$ refers to the number of spanning trees used in CASE x.

CASE 2. All 2^{n-1} exterior links operate and at least one $(n-1)$-cube fails.

When all exterior links are good, the congruent nodes in the two C_{n-1}'s can be combined to form an $(n-1)$-cube, C'_{n-1}, in which each pair of adjacent nodes is connected by double links. Consider the C_3 shown in Figure 4.2. Figure 4.3 gives the graph representation for CASE 2 in which the congruent nodes are combined. The reliability of the C'_{n-1} is given as the lower bound on the reliability of a C_{n-1} with link reliability $p' = p^2 + 2pq$ $(= 1 - q^2)$, since the C'_{n-1} is connected when it contains at least one working spanning tree, which may have links from either or both C_{n-1}'s. Thus, we have the following expression.

$$p^{2^{n-1}} \cdot (NCF(n-1, p') - NCF(n-1, p)^2) \tag{4.14}$$

The events considered in $NCF(n-1, p')$ include the possibilities that both subcubes operate, so we subtract $NCF(n-1, p)^2$ to make CASE 2 disjoint from CASE 1. For CASE 2, a spanning tree is constructed from all the exterior links and a spanning tree in the C'_{n-1}. Since links in the C'_{n-1} are from links of the two $(n-1)$-cubes, there are $ST(n-1)\, 2^{(2^{n-1}-1)}$ possible spanning trees in the C'_{n-1} and, thus, the number of spanning trees used in CASE 2, $ST_{CASE\,2}(n)$, is $ST(n-1)\, 2^{(2^{n-1}-1)}$.

CASE 3. $2 \le i \le 2^{n-1}-1$ exterior links and one $(n-1)$-cube operate.

CASE 3 is intractable for large n, so we compute a lower bound on reliability.

CASE 3A for $i = 2^{n-1}-1$

Figure 4.4a depicts the graph representation of CASE 3A for a C_3. As an example, the figure shows an event in which links 8, 9, and 10 operate while link 11 fails. Let an *isolated node* be defined as a node in a disconnected subcube C_{n-1} that has a failed exterior link as one of its links. The C_3 is connected if one C_2 cube operates and the isolated node (in the other C_2) is connected by at least one of its interior links (links 6 and 7). There are two possibilities for the working C_{n-1} and 2^{n-1} possibilities for the failed exterior link. The probability that at least one interior link incident on the isolated node is working is $1 - q^{n-1}$. We have included the event in which the two $(n-1)$-cubes operate. To make this case disjoint with CASE 1, we subtract $NCF(n-1, p)^2$. The reliability contribution for this case is then given as:

$$2 \cdot 2^{n-1} \cdot p^{2^{n-1}-1} \cdot q \cdot (NCF(n-1, p) \cdot (1-q^{n-1}) - NCF(n-1, p)^2). \quad (4.15)$$

For CASE 3A, a spanning tree is constructed from a spanning tree in $(n-1)$-cube, $2^{n-1}-1$ exterior links, and any one of $(n-1)$ interior links connecting the isolated node to one of its neighbors. There are two possibilities for the working C_{n-1}, 2^{n-1} possibilities for the working $2^{n-1}-1$ exterior links, and $(n-1)$ possibilities for interior links of the isolated node. Thus, CASE 3A considers $ST_{CASE\,3A}(n) = ST(n-1) \cdot (n-1) \cdot 2^n$ spanning trees.

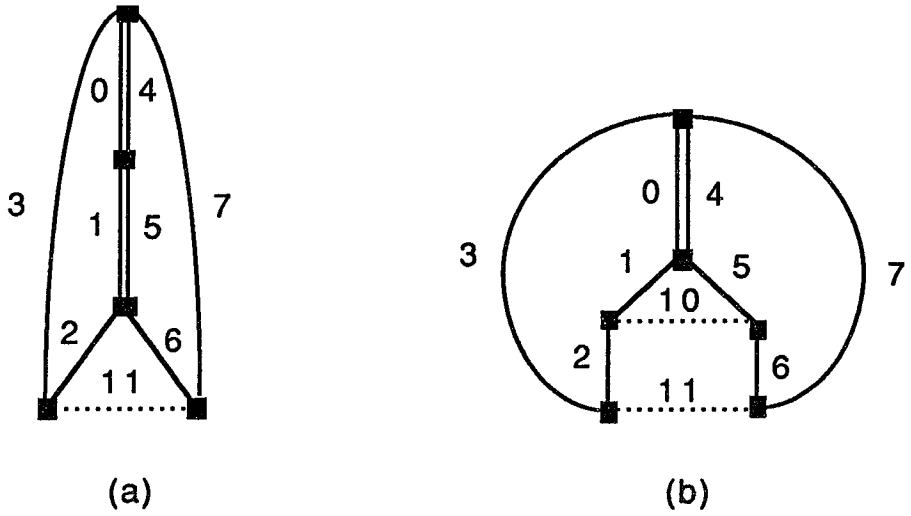


Figure 4.4 A C_3 with (a) A Faulty Exterior Link,

(b) Two Isolated 1-distant PE's, and

(c) Two Isolated PE's Different than that Given in (b)

CASE 3B for $i = 2^{n-1}-2$

In CASE 3B, the disconnected C_{n-1} includes two isolated nodes which can be at distance one or more. We consider the reliability expression for this case in two subcases based on the distance between the isolated nodes.

The two nodes are at distance 1: Figure 4.4b presents the graph model of this case for C_3 showing the event in which links 8 and 9 operate, while links 10 and 11 fail. Since the two isolated nodes are at distance 1, there is a link connecting them. First, consider the link operational. Thus, the two nodes can be merged into a node X. The C_n is connected if node X is connected by at least one of its interior links. This event has probability $1 - q^{2(n-1)-2}$. Second, consider the connecting link as failed. In this case, each of the two nodes should individually be connected by at least one of its interior links. This event has probability $(1 - q^{n-2})^2$. Again, we have included the event in which both C_{n-1}'s operate, so we subtract $NCF(n-1, p)^2$ to make this case disjoint from CASE 1. There are again two possibilities for the working C_{n-1}. The lower bound expression for this case is:

$$2 \cdot L' \cdot p^{2^{n-1}-2} \cdot q^2 \cdot (NCF(n-1, p) \cdot (1-q^{2(n-1)-2}) \cdot p + q \cdot (1-q^{n-2})^2) - NCF(n-1, p)^2)), \quad (4.16)$$

where L' is the number of possibilities that any two nodes are at distance 1 (which is given by the number of links in a C_{n-1}). For this case, a spanning tree is constructed from a spanning tree of an $(n-1)$-cube, $2^{n-1}-2$ exterior links, and an interior link connecting the two isolated nodes to one of their neighbors when the nodes are connected or two interior links connecting the nodes to their neighbors when the nodes are not connected. There are two possibilities for the working C_{n-1}, $(n-1) \cdot 2^{n-2}$ possibilities of the working $2^{n-1}-2$ links, $2(n-1)-2$ possibilities for interior links connecting the two nodes to their neighbors when the nodes are

connected, and $(n-1)^2$ possibilities for links connecting the isolated nodes to their neighbors when the nodes are not connected. Thus, this case considers $ST(n-1)\ (n^3-3n^2+2n)\ 2^{n-1}$ spanning trees.

The two nodes are at a distance of more than 1: Figure 4.4c presents the graph model of this case for C_3, showing the event in which links 8 and 10 operate, while links 9 and 11 fail. When the two isolated nodes are at distance of more than one from each other, the n-cube is connected when each of them is connected by at least one of its interior links. The probability of this event is $(1 - q^{n-1})^2$. Again, we have considered the case in which both $(n-1)$-cubes operate so we subtract $NCF(n-1, p)^2$. Again, there are two possibilities for the working C_{n-1}. The lower bound expression for this case is:

$$2 \cdot \left(\begin{bmatrix} 2^{n-1} \\ 2^{n-1}-2 \end{bmatrix} - L'\right) \cdot p^{2^{n-1}-2} \cdot q^2 \cdot (NCF(n-1, p) \cdot$$

$$(1-q^{n-1})^2 - NCF(n-1, p)^2), \qquad (4.17)$$

where $\left(\begin{bmatrix} 2^{n-1} \\ 2^{n-1}-2 \end{bmatrix} - L'\right)$ is the number of choices of two nodes at distance more than 1. For this case, a spanning tree is constructed from a spanning tree of an $(n-1)$-cube, $2^{n-1}-2$ exterior links, and an interior link for each of the two isolated nodes. There are two possibilities for the working C_{n-1}, $\left(\begin{bmatrix} 2^{n-1} \\ 2^{n-1}-2 \end{bmatrix} - L'\right)$ possibilities for the $2^{n-1}-2$ working exterior links, and $(n-1)^2$ possibilities for interior links connecting the two nodes to their neighbors, so the number of spanning trees considered in this case is $ST(n-1)\ (n-1)^2\ (2^{n-1}-n)\ 2^{n-1}$. Thus, overall, CASE 3B uses $ST_{CASE3B}(n) = ST(n-1)\ 2^{n-1}(2^{n-1}(n^2-2n+1)-n^2+n)$ spanning trees.

CASE 3C for $2 \leq i \leq 2^{n-1}-3$

The problem of obtaining an exact expression for this case becomes intractable, especially for large n. One can get a lower bound on the reliability contribution of this case, however, by considering the number of spanning trees that have not been included in the reliability evaluated so far, and take a lower bound on reliability for each of them, that is, multiply the number by $p^{N-1} \cdot q^d$, where $d = L-N+1$. Thus, a lower bound on reliability expression for CASE 3C is:

$$p^{N-1} \cdot q^d \cdot (ST(n) - [ST_{CASE1}(n) + \cdots + ST_{CASE3B}(n)]). \tag{4.18}$$

Combining all cases,

$$NCF(n, p) \geq (4.13) + (4.14) + (4.15) + (4.16) + (4.17) + (4.18). \tag{4.19}$$

Theorem 4.3. Equation (4.19) provides a lower bound on the NCF measure of an n-dimensional cube that may be evaluated in $O(n^2)$ time.

Proof: The discussion in Section 4.3.2 above establishes how Equation (4.19) gives a lower bound on NCF.

For the time complexity, observe that to compute $NCF(n, p)$, we must compute $NCF(n-1, p)$, $NCF(n-1, 1-q^2)$, and p and q raised to various powers, including $q^{2^{n-1}}$, $p^{2^{n-1}}$, and q^{n-1}. To evaluate these terms, we must compute $NCF(n-2, p)$, $NCF(n-2, 1-q^2)$, $NCF(n-2, 1-q^4)$, and p and q raised to various powers. Continuing, we see that at the ith level of recursion, we must compute $i+1$ terms of the form $NCF(n-i, y)$, where $y \in \{p, 1-q^2, 1-q^4, \cdots, 1-q^{2^{i-1}}\}$. Given the NCF's from the previous level of recursion, we find that we can compute the powers of p and q in $O(n-i)$ time and the $i+1$ NCF terms in $O(i)$ time, leading to $O(n)$ time to compute the terms at the ith level of recursion. Overall, this leads to an $O(n^2)$ time complexity to compute $NCF(n, p)$. □

4.3.3 Discussion

To show the efficiency of the proposed technique, we use the technique to solve NCF for hypercubes of dimensions 3 through 16 (containing 12 through 2^{19} links). The results show that the proposed technique generates tighter lower bounds on NCF compared to the previous method. Table 4.1 shows the comparisons of the exact reliability of C_3 (for the cases described) obtained by CAREL [74] and the lower bound results produced by the proposed algorithm. The new technique generates a tight lower bound on NCF for C_3. Table 4.2 presents the comparisons of the best known lower bounds on NCF obtained in [15] with the new lower bounds generated by our proposed technique for $n = 3, 4, \cdots, 16$. Notice that the BD technique does not generate a tight bound even for a C_3. As shown in the table, our lower bounds are tighter than the BD bounds, especially for $n > 10$ and $p < 0.9$. The table shows the sturdiness of the hypercube structure, particularly, when link reliability $p \geq 0.95$. Observe that for smaller values of p and larger values of n, the bounds deteriorate rapidly. To explain this we need to consider that NCF relates to the magnitude of p and the number of spanning trees, $ST(n)$. In particular, the reliability contribution of each spanning tree is weighted by p^{N-1}. Thus, for increasing values of n and smaller values of p, the reliability contribution of each spanning tree decreases rapidly since $N = 2^n$. On the other hand, $ST(n)$ increases superexponentially in the order of n. Thus, the NCF for n-cube can be larger or smaller than that of w-cube, for $n < w$, depending on the tradeoff between the reliability contribution of each spanning tree and $ST(n)$. The lower bounds on NCF computed by our method show this characteristic.

Table 4.1
NCF Measure for C_3 - Exact vs. Lower Bound for p = 0.9

CASE	#Spanning Trees		NCF Reliability*	
	Exact	Lower Bound	Exact	Lower Bound
1	64	64	0.898045	0.898045
2	32	32	0.066445	0.066445
3a	64	64	0.023380	0.023379
3b	160	160	0.002488	0.002487
3c	64	64	0.000306	0.000306

* The exact reliability is 0.990664, and the lower bound obtained is 0.990662

Table 4.2
Lower Bounds on NCF for C_n

n-cube	Link Reliability = 0.9		Link Reliability = 0.95	
	Bulka-Dugan	New Bound	Bulka-Dugan	New Bound
3	9.878696e-01	9.906629e-01	9.986941e-01	9.989089e-01
4	9.946773e-01	9.978844e-01	9.997803e-01	9.998912e-01
5	9.947729e-01	9.989311e-01	9.999122e-01	9.999824e-01
6	9.911984e-01	9.986355e-01	9.999155e-01	9.999919e-01
7	9.826412e-01	9.973788e-01	9.998587e-01	9.999899e-01
8	9.655845e-01	9.947654e-01	9.997205e-01	9.999806e-01
9	9.323534e-01	9.895581e-01	9.994411e-01	9.999612e-01
10	8.692828e-01	9.792253e-01	9.988826e-01	9.999223e-01
11	7.556526e-01	9.588822e-01	9.977664e-01	9.998446e-01
12	5.710109e-01	9.194552e-01	9.955378e-01	9.996893e-01
13	3.260535e-01	8.453978e-01	9.910955e-01	9.993787e-01
14	1.063109e-01	7.146974e-01	9.822703e-01	9.987578e-01
15	1.130200e-02	5.107924e-01	9.648550e-01	9.975171e-01
16	1.277352e-04	2.609089e-01	9.309451e-01	9.950403e-01

4.4 Methods for Bounding the Probabilistic 2CF Measure

4.4.1 An Overview

The lower bound on the probabilistic 2CF measure is obtained by using the reliability polynomial concepts discussed in Section 4.1. However, the method is only polynomial in the order of the number of components of the n-cube, and, hence, exponential in the order of cube dimension n. Furthermore, reliability bounds generated by polynomial-based algorithms are not tight. In what follows, methods to obtain the lower bounds on 2CF are discussed using shortest length paths.

We first discuss a lower bound on 2CF measure using shortest paths. Let (s, t) be a pair of nodes at distance n in a w-cube, where $n \leq w$. We will consider only shortest paths between s and t to obtain a lower bound, so we need only discuss the n-subcube containing s and t. To simplify the discussion, consider s and t as nodes at diameter distance in an n-cube, C_n. Without loss of generality, consider the addresses for s and t to be 0 and 2^n-1, respectively. C_n contains $n!$ (s, t) paths [70], each of which traverses $n+1$ nodes and n links. For 2CF under the node failure model, we assume that nodes s and t are always perfect. 2CF measure can be computed by first enumerating the $n!$ (s, t) paths, treating each path description as a product term of Boolean variables, and then use any SDP technique [20,33,74,85] to transform the sum-of-product terms into the analogous reliability expression. The number of paths for the cube grows superpolynomially with the number of nodes and links in C_n; hence, it is not efficient to compute 2CF measure for large n using the above method. Alternatively, computing a lower bound on 2CF measure offers the possibility of obtaining an important insight into the value of 2CF at a much better computational cost.

Of the $n!$ (s, t) paths, n of these are node and link disjoint; this is true for any (s, t) pair because C_n is both node and link symmetric. A lower bound on 2CF is easily computed by considering the n disjoint paths utilizing a method used to obtain the reliability of a series-parallel system [20]. Let $2CF_1(n, r, p)$ be the lower bound on 2CF for a C_n with node (link) reliability r (p) by considering only its n disjoint paths. Then, for perfect source and terminal nodes we obtain:

$$2CF_1(n, r, p) = 1 - (1 - r^{n-1}p^n)^n. \tag{4.20}$$

The model is straightforward, but the accuracy rapidly deteriorates as n increases.

Latifi [47] derived a lower bound on 2CF by considering nodes s and t to be at distance $n \leq w$. They used w disjoint paths in the C_w comprising n paths of length n and $w-n$ paths of length $n+2$, and assumed that the nodes are perfect. Ahmed and Trahan [3] provided a technique to approximate 2CF for the node failure case (perfect links) by considering all shortest paths. Their method runs in time exponential in the distance of the (s, t) nodes, however, and so is not efficient. In Section 4.4.2, we provide a polynomial time algorithm which obtains a tighter bound on terminal reliability for an n-cube compared with Equation (4.20).

4.4.2 Proposed Bounds on 2CF

This section develops two methods, namely, a Boolean technique and a two-cube (TC) approach, for evaluating lower bounds on 2CF. The methods consider node and/or link failures. Subsection 4.4.2.1 discusses a Boolean algebraic approach for a lower bound on 2CF for larger values of node reliability r and link reliability p, while Subsection 4.4.2.2 presents the TC approach that offers a tighter bound for smaller values of r and p.

4.4.2.1 Boolean Approach

To improve the lower bounds on 2CF given in Equation (4.20), we consider $\alpha \cdot n$ shortest paths (where $\alpha = \lceil n/2-1 \rceil$ or $n-2$; refer to Lemmas 4.1 and 4.2), selected and ordered using a path generation algorithm described below, and then use a Boolean related disjoint products technique to compute the reliability values. The $\alpha \cdot n$ selected paths include the n disjoint paths used for generating Equation (4.20), so our lower bounds are tighter. Furthermore, the proposed method uses the paths to analytically calculate the reliability without enumerating them. Consider the $\alpha \cdot n$ paths to be in α groups, each comprising n paths. For $i = 1, 2, \cdots, \alpha$, let $G_i = \{P_{i,j} \mid j = 1, 2, \cdots, n\}$ denote the ith group of paths. In the following, we present an algorithm that enumerates the $\alpha \cdot n$ paths in C_n that satisfy certain properties which will be useful for reliability evaluation.

A. Path Generation Algorithm

Any shortest path from node 0 to node 2^n-1 traverses n links in n different dimensions. Number the dimensions by 0 to $n-1$. Let $P_{i,j} = d_0 d_1 \cdots d_{n-1}$, where $\{d_0, d_1, \cdots, d_{n-1}\} = \{0, 1, \cdots, n-1\}$, denote the path from node 0 to node 2^n-1 obtained by first traversing a link in dimension d_0, then traversing a link in dimension d_1, $\cdots$, then traversing a link in dimension d_{n-1}.

Algorithm PG

1) $P_{1,1} = 0\ 1\ 2\ 3\ 4 \cdots (n-1)$

for $j = 2$ to n **do begin**

Generate path $P_{1,j}$ by traversing dimensions in the order given by a left rotate by one position of $P_{1,j-1}$.

end;

2) **for** $i = 2$ to α **do begin**

for $j = 1$ to n **do begin**

Set the first traversed dimension of $P_{i,j}$ to be the same as that of $P_{1,j}$.

Set the last $i-1$ traversed dimensions of $P_{i,j}$ to be the same as those in $P_{i-1,j}$. Generate the remaining $n-i$ dimensions (that is, the $2nd$ to $(n-i+1)th$ traversed dimensions) by a left rotate by one position of the dimensions in the same positions in $P_{i-1,j}$.

end;

end;

Lemma 4.1. For $\alpha = \lceil n/2-1 \rceil$, Algorithm PG generates $\lceil n/2 -1 \rceil \cdot n$ paths satisfying the following properties, where $1 \le i, k \le \lceil n/2 -1 \rceil$ and $1 \le j, l \le n$ and terminal nodes s and t are excluded.

Property N1. Paths $P_{i,j}$ and $P_{i,l}$ are node and link disjoint, for $j \ne l$.

Property N2. Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common nodes and links, for $i < k$.

Property N3. Paths $P_{i,j}$ and $P_{k,l}$ are node and link disjoint, for $j \ne l$.

Proof: Given in Appendix B. □

Note: For n even, we can add a set of $n/2$ paths, $G'_{\alpha+1} = \{P_{n/2,1}, P_{n/2,2}, \cdots, P_{n/2,n/2}\}$, that satisfy a similar list of properties. These properties ($N4$ - $N6$) are stated and proved in Appendix B. We will include this extra half-size group of paths for computing 2CF for the node failure case and the node and link failure case, though we do not explicitly refer to these paths in the following sections.

Lemma 4.2. For $\alpha = n-2$, Algorithm PG generates $(n-2)\cdot n$ paths satisfying the following properties, where $1 \le i, k \le n-2$ and $1 \le j, l \le n$.

Property L1. Paths $P_{i,j}$ and $P_{i,l}$ are link disjoint, for $j \ne l$.

Property L2. Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common links, for $i < k$.

Property L3. Paths $P_{i,j}$ and $P_{k,l}$ are link disjoint, for $j \ne l$.

Proof: Given in Appendix B. □

Lemma 4.1 applies to the paths evaluated for cases in which both nodes and links may fail or in which only nodes may fail and links are perfectly reliable. Lemma 4.2 applies to the paths evaluated for the case in which only links may fail and nodes are perfectly reliable.

Example 4.3. To illustrate Algorithm PG and Lemmas 4.1 and 4.2, consider a C_6 with source and terminal nodes $s = 0$ and $t = 63$, respectively. Algorithm PG produces $\alpha\cdot n$ paths grouped and ordered as shown in Table 4.3. We label the nodes by their decimal equivalent numbers. Paths $P_{i,j}$ are shown using dimensions, node labels, and links. A link (i, j) refers to the link between nodes i and j. For Lemma 4.1 (Lemma 4.2), $\alpha = 2$ ($\alpha = 4$), so paths $P_{1,1}$ through $P_{3,3}$ (groups G_1, G_2, and half of G_3) illustrate Lemma 4.1, while paths $P_{1,1}$ through $P_{4,6}$ (groups G_1 through G_4) illustrate Lemma 4.2.

Table 4.3
Results for Example 4.3

Group G_i	Path $P_{i,j}$	Dimensions in $P_{i,j}$	Nodes in $P_{i,j}$	Links in $P_{i,j}$
G_1	$P_{1,1}$	0,1,2,3,4,5	1,3,7,15,31	(0,1),(1,3),(3,7),(7,15),(15,31),(31,63)
	$P_{1,2}$	1,2,3,4,5,0	2,6,14,30,62	(0,2),(2,6),(6,14),(14,30),(30,62),(62,63)
	$P_{1,3}$	2,3,4,5,0,1	4,12,28,60,61	(0,4),(4,12),(12,28),(28,60),(60,61),(61,63)
	$P_{1,4}$	3,4,5,0,1,2	8,24,56,57,59	(0,8),(8,24),(24,56),(56,57),(57,59),(59,63)
	$P_{1,5}$	4,5,0,1,2,3	16,48,49,51,55	(0,16),(16,48),(48,49),(49,51),(51,55),(55,63)
	$P_{1,6}$	5,0,1,2,3,4	32,33,35,39,47	(0,32),(32,33),(33,35),(35,39),(39,47),(47,63)
G_2	$P_{2,1}$	0,2,3,4,1,5	1,5,13,29,31	(0,1),(1,5),(5,13),(13,29),(29,31),(31,63)
	$P_{2,2}$	1,3,4,5,2,0	2,10,26,58,62	(0,2),(2,10),(10,26),(26,58),(58,62),,(62,63)
	$P_{2,3}$	2,4,5,0,3,1	4,20,52,53,61	(0,4),(4,20),(20,52),(52,53),(53,61),(61,63)
	$P_{2,4}$	3,5,0,1,4,2	8,40,41,43,59	(0,8),(8,40),(40,41),(41,43),(43,59),(59,63)
	$P_{2,5}$	4,0,1,2,5,3	16,17,19,23,55	(0,16),(16,17),(17,19),(19,23),(23,55),(55,63)
	$P_{2,6}$	5,1,2,3,0,4	32,34,38,46,47	(0,32),(32,34),(34,38),(38,46),(46,47),(47,63)
G_3	$P_{3,1}$	0,3,4,2,1,5	1,9,25,29,31	(0,1),(1,9),(9,25),(25,29),(29,31),(31,63)
	$P_{3,2}$	1,4,5,3,2,0	2,18,50,58,62	(0,2),(2,18),(18,50),(50,58),(58,62),(62,63)
	$P_{3,3}$	2,5,0,4,3,1	4,36,37,53,61	(0,4),(4,36),(36,37),(37,53),(53,61),(61,63)
	$P_{3,4}$	3,0,1,5,4,2	8,9,11,43,59	(0,8),(8,9),(9,11),(11,43),(43,59),(59,63)
	$P_{3,5}$	4,1,2,0,5,3	16,18,22,23,55	(0,16),(16,18),(18,22),(22,23),(23,55),(55,63)
	$P_{3,6}$	5,2,3,1,0,4	32,36,44,46,47	(0,32),(32,36),(36,44),(44,46),(46,47),(47,63)
G_4	$P_{4,1}$	0,4,3,2,1,5	1,17,25,29,31	(0,1),(1,17),(17,25),(25,29),(29,31),(31,63)
	$P_{4,2}$	1,5,4,3,2,0	2,34,50,58,62	(0,2),(2,34),(34,50),(50,58),(58,62),(62,63)
	$P_{4,3}$	2,0,5,4,3,1	4,5,37,53,61	(0,4),(4,5),(5,37),(37,53),(53,61),(61,63)
	$P_{4,4}$	3,1,0,5,4,2	8,10,11,43,59	(0,8),(8,10),(10,11),(11,43),(43,59),(59,63)
	$P_{4,5}$	4,2,1,0,5,3	16,20,22,23,55	(0,16),(16,20),(20,22),(22,23),(23,55),(55,63)
	$P_{4,6}$	5,3,2,1,0,4	32,40,44,46,47	(0,32),(32,40),(40,44),(44,46),(46,47),(47,63)

B. Generalized Lower Bound Expression

Let $F_{i,j}$ denote the event that path $P_{i,j}$ is operational. The event F that at least one (s, t) path out of $\alpha \cdot n$ works is given by:

$$F = \bigcup_{i,j} F_{i,j} ,$$

where $1 \le i \le \alpha$ and $1 \le j \le n$. Assume $B_{i,j}$ is a Boolean product term corresponding to $F_{i,j}$. Then,

$$B = \sum_{i,j} B_{i,j} , \tag{4.21}$$

where $1 \le i \le \alpha$ and $1 \le j \le n$, is the Boolean expression corresponding to F. By conservative modification, the SDP form for Equation (4.21) is obtained as:

$$\begin{aligned} B(disjoint) &= B_{1,1} + B_{1,2}\,\overline{B_{1,1}} + \cdots + [B_{1,n}\,\overline{B_{1,1}} \cdots \overline{B_{1,n-1}}] + \cdots \\ B(disjoint) &= B_{1,1} + B_{1,2}\,B_{1,1} + \cdots + [B_{1,n}\,B_{1,1} \cdots B_{1,n-1}] + \cdots \\ &\quad + [B_{\alpha,n}\,\overline{B_{1,1}} \cdots \overline{B_{1,n}} \cdots \overline{B_{\alpha,n-1}}]. \end{aligned} \tag{4.22}$$

(Term $\overline{B_{i,j}}$ corresponds to the event $\overline{F_{i,j}}$ that path $P_{i,j}$ is not operational.) Note, the ith n terms in Equation (4.22) correspond to the event that some path in group G_i is working and all paths in groups $G_1, \cdots, G_{i-1}$ are failed. Use any Boolean technique to generate an equivalent exclusive and mutually disjoint expression for Equation (4.22). In this form, a logical expression has a one-to-one correspondence with the probability expression. For node (link) reliability r (p), let $R_i(n, r, p)$ denote the reliability contribution of G_i corresponding to the ith n terms in Equation (4.22). R_i and R_j describe the probability of disjoint events for $i \neq j$. The reliability contribution of the $\alpha \cdot n$ paths, $2CF_2(n, r, p)$, is then given as:

$$2CF_2(n, r, p) = \sum_{i=1}^{\alpha} R_i(n, r, p). \tag{4.23}$$

We now describe the computation of the R_i's, starting with R_1.

(a) For $i = 1$

Since node failures are independent from link failures, we obtain the probability that a path $P_{1,j}$ is operational as $\mathbf{Pr}(F_{1,j}) = r^{n-1} \cdot p^n$ because all nodes and links in $P_{1,j}$ must be up. By Property $N1$, the reliability contribution of the n paths in group G_1 is

$$R_1(n, r, p) = 1 - (1-r^{n-1} \cdot p^n)^n \ .$$

The result for R_1 is the same as that given for $2CF_1$ in Equation (4.20). It is, thus, obvious that the bound generated by Equation (4.23) is always tighter as it contains Equation (4.20).

(b) For $2 \leq i \leq \alpha$

The reliability contribution of the jth path in the ith group of n terms in Equation (4.22), $[B_{i,j}(\overline{B_{1,1}} \cdots \overline{B_{1,n}}) \cdots (\overline{B_{i-1,1}} \cdots \overline{B_{i-1,n}})(\overline{B_{i,1}} \cdots \overline{B_{i,j-1}})]$, is computed as:

$$RP_{i,j}(n, r, p) = \mathbf{Pr}(F_{i,j}) \cdot \mathbf{Pr}(\overline{F_{1,1}} \cdots \overline{F_{1,n}} \cdots \overline{F_{i,1}} \cdots \overline{F_{i,j-1}} \mid F_{i,j}). \quad (4.24)$$

Equation (4.24) specifies the reliability contribution of a path $P_{i,j}$. Note,

$$R_i(n, r, p) = \sum_{j=1}^{n} RP_{i,j}(n, r, p), \quad (4.25)$$

since $RP_{i,j}$ and $RP_{i,k}$ describe the probability of disjoint events for $j \neq k$. By properties $N1$ through $N3$, Equation (4.24) is evaluated by taking the product of subterms given in expressions (4.26) through (4.29) below.

$$\mathbf{Pr}(F_{i,j}) \quad (4.26)$$

$$\mathbf{Pr}(\overline{F_{i-1,j}}\ \overline{F_{i-2,j}} \cdots \overline{F_{1,j}} \mid F_{i,j}) \quad (4.27)$$

$$\mathbf{Pr}(\overline{F_{i,j-1}}\ \overline{F_{i-1,j-1}} \cdots \overline{F_{1,j-1}} \mid F_{i,j}) \cdots \mathbf{Pr}(\overline{F_{i,1}}\ \overline{F_{i-1,1}} \cdots \overline{F_{1,1}} \mid F_{i,j}) \quad (4.28)$$

$$\mathbf{Pr}(\overline{F_{i-1,j+1}}\ \overline{F_{i-2,j+1}} \cdots \overline{F_{1,j+1}} \mid F_{i,j}) \cdots \mathbf{Pr}(\overline{F_{i-1,n}}\ \overline{F_{i-2,n}} \cdots \overline{F_{1,n}} \mid F_{i,j}) \quad (4.29)$$

Observe that expression (4.26) corresponds to path $P_{i,j}$ up, expression (4.27) corresponds to path $P_{k,j}$ down, where $1 \leq k \leq i-1$, given that $P_{i,j}$ is up, expression (4.28) corresponds to path $P_{k,l}$ down, where $1 \leq k \leq i$, $1 \leq l \leq j-1$, given that $P_{i,j}$ is up, and expression (4.29) corresponds to path $P_{k,l}$ down, where $1 \leq k \leq i-1$, $j+1 \leq l \leq n$, given that $P_{i,j}$ is up. By property $N3$, each $F_{k,l}$ in expressions (4.28) and (4.29) has no elements in common with $F_{i,j}$, so these conditional probabilities can be replaced by unconditional probabilities. For $b \neq d$, $F_{a,b}$ and $F_{c,d}$ in expressions (4.28) and (4.29) have no common elements. In the following, we evaluate expressions (4.26) through (4.29).

Expression (4.26): Expression (4.26) is directly computed as

$$\mathbf{Pr}(F_{i,j}) = r^{n-1} \cdot p^{n}. \tag{4.30}$$

Expression (4.27): To help compute expression (4.27), let $P_{a,b} - P_{c,d}$ denote the portion of path $P_{a,b}$ remaining after any nodes and links in common with $P_{c,d}$ are removed. For any $k, l < i$ and $k \neq l$, $P_{k,j} - P_{i,j}$ and $P_{l,j} - P_{i,j}$ are node and link disjoint, so expression (4.27) is computed as:

$$\mathbf{Pr}(\overline{F_{i-1,j}} \mid F_{i,j}) \cdot \mathbf{Pr}(\overline{F_{i-2,j}} \mid F_{i,j}) \cdots \mathbf{Pr}(\overline{F_{1,j}} \mid F_{i,j}).$$

By property $N2$ and for $v = 1, 2, \cdots, i-1$, each $\overline{F_{v,j}}$ has $v+1$ common nodes and links with $F_{i,j}$. Thus,

$$\mathbf{Pr}(\overline{F_{v,j}} \mid F_{i,j}) = 1 - r^{n-2-v} \cdot p^{n-1-v}.$$

Solving all terms of the expression for $v = 1, 2, \cdots, i-1$, the reliability contribution of expression (4.27) is given as:

$$\prod_{v=1}^{i-1} (1 - r^{n-2-v} \cdot p^{n-1-v}). \tag{4.31}$$

Expression (4.28): Let $MG_i(k, r, p)$ denote the kth probability term in the unconditional probability corresponding to expression (4.28) with node (link) reliability r (p), that is

$$MG_i(k, r, p) = \mathbf{Pr}(\overline{F_{i,k}}\ \overline{F_{i-1,k}}\ \cdots\ \overline{F_{1,k}})$$

for $1 \le k \le j-1$. By properties $N1$ through $N3$, $MG_i(k, r, p) = MG_i(g, r, p)$ for $1 \le k, g \le j-1$, so for simplicity, we let $MG_i(r, p)$ denote $MG_i(k, r, p)$ for any $1 \le k \le j-1$. Consequently, expression (4.28) is equal to

$$(MG_i(r, p))^{j-1}. \tag{4.32}$$

For $i = 1$, we directly compute $MG_1(r, p) = \mathbf{Pr}(\overline{F_{1,k}})$. Thus, letting $\Phi_g(r, p) = (1 - r^{g-1}p^g)$,

$$MG_1(r, p) = \Phi_n(r, p).$$

When $i = 2$, $MG_2(r, p) = \mathbf{Pr}(\overline{F_{1,k}}\ \overline{F_{2,k}})$. By Property $N2$, $F_{1,k}$ and $F_{2,k}$ have two nodes (links) in common. Hence, using a Boolean identity $\overline{abA}\ \overline{abB} = \overline{ab} + abA\ \overline{B}$, and letting $\Theta_g(x) = (1 - x^g)$,

$$MG_2(r, p) = \Theta_2(rp) + (rp)^2\Phi_{n-2}(r, p)^2.$$

For $i > 2$, we use Property $N2$ and recursively utilize the Boolean identity $\overline{abA}\ \overline{abB} = \overline{ab} + abA\ \overline{B}$ to obtain:

$$MG_i(r, p) = \Theta_2(rp) + (rp)^2\Phi_{n-2}(r, p)\ (\Theta_1(rp) + rp\,\Phi_{n-3}(r, p)\ (\ \cdots$$

$$(\Theta_1(rp) + rp\,\Phi_{n-(i-1)}(r, p)\ (\Theta_1(rp) + rp\,\Phi_{n-i}(rp)^2))\ \cdots\ )). \tag{4.33}$$

Expression (4.29): Expression (4.29) is computed similarly as

$$(MG_{i-1}(r, p))^{n-j}. \tag{4.34}$$

Taking the product of expressions (4.30), (4.31), (4.32), and (4.34) to express the product of expressions (4.26) through (4.29), the reliability contribution of a path $P_{i,j}$ described in Equation (4.24) may be taken as:

$$RP_{i,j}(r,p) = r^{n-1}\cdot p^n \prod_{v=1}^{i-1}(1 - r^{n-2-v}\cdot p^{n-1-v}) \cdot$$

$$(MG_i(r,p))^{j-1} \cdot (MG_{i-1}(r,p))^{n-j}. \tag{4.35}$$

Thus, by Equation (4.25), the reliability contribution of the n paths in group G_i is obtained as:

$$R_i(n,r,p) = r^{n-1}\cdot p^n \prod_{v=1}^{i-1}(1 - r^{n-2-v}\cdot p^{n-1-v})$$

$$\sum_{j=1}^{n}[(MG_i(r,p))^{j-1}\,(MG_{i-1}(r,p))^{n-j}]. \tag{4.36}$$

Similarly, the reliability contribution of the $n/2$ paths in group $G'_{\alpha+1}$, for n even, is computed as:

$$R_{\alpha+1}(n,r,p) = r^{n-1}\cdot p^n \prod_{v=1}^{\alpha}(1 - r^{n-2-v}\cdot p^{n-1-v})$$

$$\sum_{j=1}^{n/2}[(MG_{\alpha+1}(r,p))^{j-1}\,(MG_{\alpha}(r,p))^{n-j}].$$

Theorem 4.4. Equation (4.23) obtains a lower bound on the terminal reliability in a hypercube for a source and terminal node at distance n. This bound may be evaluated in $O(n^2)$ time. Further, this bound is tighter than the bound based on disjoint shortest paths.

Proof: Section 4.4.2.1 establishes how Equation (4.23) gives a lower bound on 2CF, given that the paths have the properties described in Lemmas 4.1 and 4.2. Compared to the bound based on n disjoint shortest paths (Equation (4.20)), Equation (4.23) is clearly tighter as it incorporates these n paths in the $\alpha \cdot n$ paths it evaluates.

For time complexity, Equation (4.23) prescribes a sum of $R_1(n, r, p)$, $R_2(n, r, p)$, $\cdots$, $R_\alpha(n, r, p)$. Consider Equation (4.36) for $R_i(n, r, p)$. We can precompute $\Phi_k(r, p)$, for all $1 \leq k \leq n$, in $O(n)$ time. Using these, we can compute each $MG_i(r, p)$ in $O(i)$ time, for $O(n^2)$ time for all $MG_i(r, p)$'s. Next, compute $(MG_i(r, p))^k$, for all $1 \leq k \leq n$, in $O(n)$ time. Using these precomputed values, each $R_i(n, r, p)$ can be computed in $O(n)$ time. Since $\alpha = O(n)$, Equation (4.23) may be evaluated in $O(n^2)$ time. □

4.4.2.2 Two-cube Model

One can view a C_n as constructed of two C_{n-1}'s that are connected by 2^{n-1} links. Call these links as *exterior* links, and the links within each C_{n-1} as *interior* links. Let $2CF_3(n, r, p)$ denote a lower bound function on 2CF for a C_n with node (link) reliability r (p). We will compute $2CF_3(n, r, p)$ for a C_n from a recursive computation of a lower bound on the 2CF of its component C_{n-1}'s. Nodes 0 and $2^n - 1$ are the source and terminal nodes of the C_n. Let $s_1 = 0$ and $t_1 = 2^{n-1}-1$ be the source and terminal nodes of one C_{n-1}, and $s_2 = 2^{n-1}$ and $t_2 = 2^n-1$ be the source and terminal nodes of the other C_{n-1}. For the recursive lower bound computation, recursively decompose C_n as follows. Let $2CF_3(n-1, r, p)$ be the probability of a working path in one C_{n-1} between s_1 and t_1, and also in the other C_{n-1} between s_2 and t_2. Note, s_1 and t_1, and s_2 and t_2 are at diameter distance $n-1$ in

their corresponding C_{n-1}'s. Connect s_1 and s_2 by an exterior link and t_1 and t_2 by an exterior link such that these links each have reliability p (refer to Figure 4.5). We ignore the other $2^{n-1}-2$ exterior links in the lower bound computation. Let A (B) denote the event that there exists at least one working path between s_1 and t_1 (s_2 and t_2). Similarly, let C (D) denote the event that the link connecting nodes s_1 and s_2 (t_1 and t_2) is working, and let E (F) denote the event that node t_1 (s_2) is working. Thus, (AED) or (CFB) represents the event in which a working (s_1, t_2) path exists, the probability of which is given as:

$$\mathbf{Pr}(\text{AED} + \text{CFB}) = \mathbf{Pr}(\text{AED}) + \mathbf{Pr}(\text{CFB}) \cdot (1 - \mathbf{Pr}(\text{AED})). \qquad (4.37)$$

Since events A through F are independent, $\mathbf{Pr}(\text{AED}) = \mathbf{Pr}(\text{A}) \cdot \mathbf{Pr}(\text{E}) \cdot \mathbf{Pr}(\text{D})$. Similarly, $\mathbf{Pr}(\text{CFB}) = \mathbf{Pr}(\text{C}) \cdot \mathbf{Pr}(\text{F}) \cdot \mathbf{Pr}(\text{B})$. Note, $\mathbf{Pr}(\text{A})$ and $\mathbf{Pr}(\text{B})$ are each $2CF_3(n-1, r, p)$, and $\mathbf{Pr}(\text{E})$ and $\mathbf{Pr}(\text{F})$ ($\mathbf{Pr}(\text{C})$ and $\mathbf{Pr}(\text{D})$) are the node reliability r (link reliability p). Thus, the lower bound on 2CF for a C_n is obtained recursively from the bounds on 2CF for lower dimension cubes. As the base case for this recursion, we use the exact 2CF for a 3-cube.

Theorem 4.5. For a C_n with node (link) reliability r (p), a two-cube model based lower bound on 2CF is:

$$2CF_3(n, r, p) = 2rp \cdot 2CF_3(n-1, r, p) - (rp\,2CF_3(n-1, r, p))^2. \qquad (4.38)$$

This expression may be evaluated in $O(n)$ time.

Proof: From Equation (4.37) and the following discussion, substitute $2CF_3(n-1, r, p)$, link reliability p, and node reliability r for $\mathbf{Pr}(\text{A})$ and $\mathbf{Pr}(\text{B})$, $\mathbf{Pr}(\text{C})$ and $\mathbf{Pr}(\text{D})$, and $\mathbf{Pr}(\text{E})$ and $\mathbf{Pr}(\text{F})$, respectively.

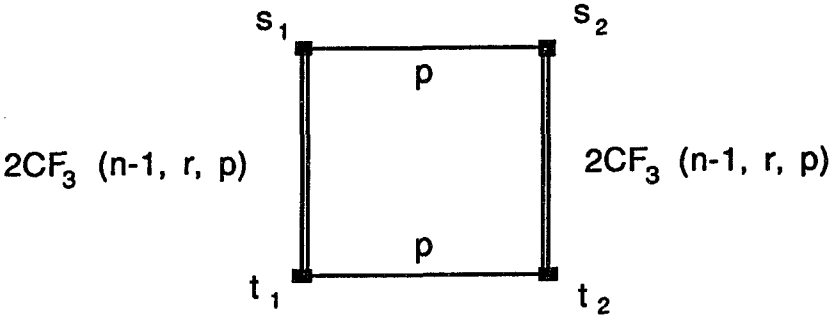


Figure 4.5 A Two-cube (TC) Model of a C_n

For the time complexity, $2CF_3(n, r, p)$ may be computed in constant time plus the time required to compute $2CF_3(n-1, r, p)$. Clearly, this leads to $O(n)$ time complexity. □

Corollary. A tighter lower bound on terminal reliability for C_n with node reliability r and link reliability p, $2CF(n, r, p)$, is computed as:

$$2CF(n, r, p) = \max \{ 2CF_2(n, r, p), 2CF_3(n, r, p)\}. \tag{4.39}$$

By Theorems 4.4 and 4.5, this bound may be computed in $O(n^2)$ time.

4.4.3 Illustrations

Example 4.4. This example shows how to compute a lower bound on 2CF for the link failure case, that is, nodes are perfect ($r = 1$). Consider a C_6 with link reliability $p = 0.6$ and $\alpha = 4$. We compute $R_1(6, 1, p) = 1 - (1 - p^6)^6$, $MG_1(1, p) = \Phi_6(1, p) = 1 - p^6$, and $MG_2(1, p) = \Theta_2(p) + p^2\Phi_{6-2}(1,p)^2 = (1 - p^2) + p^2(1 - p^{6-2})^2$. By Equation (4.33), we obtain: $MG_3(1, p) = \Theta_2(p) + p^2\Phi_{6-2}(1, p)((1-p)+p\,\Phi_{6-3}(1, p)^2)$, and $MG_4(1, p) = \Theta_2(p) + p^2\Phi_{6-2}(1, p)((1-p)+p\,\Phi_{6-3}(1, p)((1-p) + p\,\Phi_{6-4}(1, p)^2))$. Utilizing Equation (4.36), we get the reliability contributions, $R_i(6, 1, p)$, as:

$$R_2(6, 1, p) = p^6 \prod_{\nu=1}^{2-1}(1-p^{6-1-\nu}) \sum_{j=1}^{n}[(MG_2(1, p))^{j-1} (MG_1(1, p))^{6-j}].$$

$$R_3(6, 1, p) = p^6 \prod_{\nu=1}^{3-1}(1-p^{6-1-\nu}) \sum_{j=1}^{n}[(MG_3(1, p))^{j-1} (MG_2(1, p))^{6-j}].$$

$$R_4(6, 1, p) = p^6 \prod_{\nu=1}^{4-1}(1-p^{6-1-\nu}) \sum_{j=1}^{n}[(MG_4(1, p))^{j-1} (MG_3(1, p))^{6-j}].$$

Substituting $p = 0.6$, one gets $MG_1 = 0.953344$, $MG_2 = 0.912735$, $MG_3 = 0.880897$, and $MG_4 = 0.860521$. We also obtain $R_1(6, 1, 0.6) = 0.249246$, $R_2(6, 1, 0.6) =$

0.172569, $R_3(6, 1, 0.6) = 0.110934$, and $R_4(6, 1, 0.6) = 0.061212$, and hence Equation (4.23) gives $2CF_2(6, 1, 0.6) = 0.593961$. On the other hand, given $2CF(5, 1, 0.9) = 0.639749$, Equation (4.38) gives $2CF_3(6, 1, 0.6) = 0.620358$, thus, the lower bound on 2CF is obtained by Equation (4.39) as $2CF(6, 1, 0.6) = 0.620358$ which gives a tighter lower bound compared to 0.249246 computed by Equation (4.20).

Example 4.5. In this example, we show how to compute a lower bound on 2CF for the node and link failure case. Consider a C_6 with node reliability $r = 0.9$ and link reliability $p = 0.95$, and $\alpha = 2$. Note, for even cube dimension, $R_{\alpha+1}$ is also computed. We obtain $R_1(6, r, p) = 1 - (1 - r^5p^6)^6$, $MG_1(r, p) = \Phi_6(r, p)$, $MG_2(r, p) = \Theta_2(rp) + (rp)^2\Phi_{6-2}(r, p)^2$, and $MG_3(r, p) = \Theta_2(rp) + (rp)^2\Phi_{6-2}(r, p)((1-rp)+rp\,\Phi_{6-3}(r, p)^2)$. Utilizing Equation (4.36), we get:

$$R_2(6, r, p) = (r^5p^6)\prod_{\nu=1}^{2-1}(1-r^{6-2-\nu}p^{6-1-\nu}) \sum_{j=1}^{n}[(MG_2(r, p))^{j-1}\ (MG_1(r, p))^{6-j}].$$

$$R_3(6, r, p) = (r^5p^6)\prod_{\nu=1}^{2}(1-r^{6-2-\nu}p^{6-1-\nu}) \sum_{j=1}^{3}[(MG_3(r, p))^{j-1}\ (MG_2(r, p))^{6-j}].$$

Substituting $r = 0.9$ and $p = 0.95$ in Equation (4.23), one gets $2CF_2(6, 0.9, 0.95) = 0.997762$. On the other hand, given $2CF(5, 0.9, 0.95) = 0.995640$, Equation (4.38) gives $2CF_3(6, 0.9, 0.95) = 0.977880$ and hence, Equation (4.39) computes $2CF(6, 0.9, 0.95) = 0.997762$. Note, using Equation (4.20), the previous lower bound on 2CF is 0.967145.

4.4.4 Discussion

Table 4.4 shows the lower bound on 2CF obtained by Equation (4.39) compared with the previous bound computed by Equation (4.20). The table assumes node failure rate $\gamma = 250{,}000$ FITS and perfect links. The table gives lower bounds

Table 4.4
Lower Bounds on 2CF for C_n
(PE Failure Rate γ = 250,000 FITS).

t*	6-cube		10-cube		16-cube	
	Previous	New	Previous	New	Previous	New
400	0.996289	0.999882	0.994584	0.999999	0.982398	1.0
800	0.936203	0.992947	0.835824	0.997336	0.558292	0.992547
1200	0.780169	0.946492	0.501277	0.923955	0.163675	0.878612
1600	0.582086	0.830946	0.241975	0.772962	0.038931	0.759296
2000	0.401843	0.679244	0.105698	0.612820	0.008813	0.586724
2400	0.263919	0.554084	0.044259	0.443028	0.001973	0.377941
2800	0.168044	0.431432	0.018212	0.286174	0.000440	0.189971
3200	0.104983	0.321827	0.007441	0.166126	0.000098	0.074191
3600	0.064830	0.231203	0.003031	0.088137	0.000022	0.023832
4000	0.039753	0.160902	0.001233	0.043622	0.000005	0.006747

* Time t is in hours.

for a 6-cube, 10-cube, and 16-cube for various mission times of the systems. The new lower bounds significantly improve on the previous bounds, especially for larger cubes and less reliable nodes (increased operation times). Table 4.5 presents the results allowing both node and link failures. We use a link failure rate, λ, 1/100th that of the node failure rate. Comparing Tables 4.4 and 4.5, we notice that for the given node and link failure rates, link failures in the system give insignificant reduction on the 2CF of the cubes. This observation is further supported by the results shown in Figures 4.6 and 4.7. Figure 4.6 depicts the 2CF vs. dimension for cubes of dimensions up to 16 for node, and both node and link failure models with various working times. We have computed exact 2CF measure for 2-cube and 3-cube. Note, a lower bound on 2CF for C_n is computed from the selected $\alpha \cdot n$ paths and their node and/or link reliabilities. On the one hand, for increasing values of n, the reliability contribution of each path decreases. On the other hand, for increasing values of n, more paths are evaluated. Thus, a 2CF for a C_k can be larger or smaller than that of a C_g for $k < g$ depending on the tradeoff between the number of paths and the reliability of each path for given node and/or link reliabilities. For t = 1000 hours, the 2CF of C_8 has the largest value. For t = 2000 hours, on the other hand, larger cubes have less 2CF's than smaller cubes. Figure 4.7 depicts the reliability performance vs. time for an 8-cube under node only, link only, and both node and link failure assumptions. As expected, for perfect nodes and link reliability $p \geq 0.988$, the 8-cube is very reliable. The figure also shows that the 2CF of an 8-cube in the node failure and node and link failure cases is closely dependent on the system mission time.

Table 4.5
Lower Bounds on 2CF for C_n
(γ = 250,000 FITS, and λ = 2,500 FITS)

t*	6-cube		10-cube		16-cube	
	Previous	New	Previous	New	Previous	New
400	0.996079	0.999873	0.994203	0.999999	0.981069	1.0
800	0.933499	0.992500	0.829271	0.997039	0.546485	0.991444
1200	0.773323	0.943684	0.490555	0.918417	0.156603	0.875661
1600	0.572693	0.823976	0.233585	0.766805	0.036560	0.753419
2000	0.392295	0.672836	0.100786	0.604436	0.008138	0.576824
2400	0.255699	0.546035	0.041731	0.432742	0.001792	0.365283
2800	0.161631	0.422562	0.016990	0.276109	0.000394	0.179132
3200	0.100277	0.313056	0.006871	0.158084	0.000086	0.068057
3600	0.061512	0.223267	0.002771	0.082687	0.000019	0.021302
4000	0.037474	0.154216	0.001116	0.040360	0.000004	0.005893

* Time t is in hours.

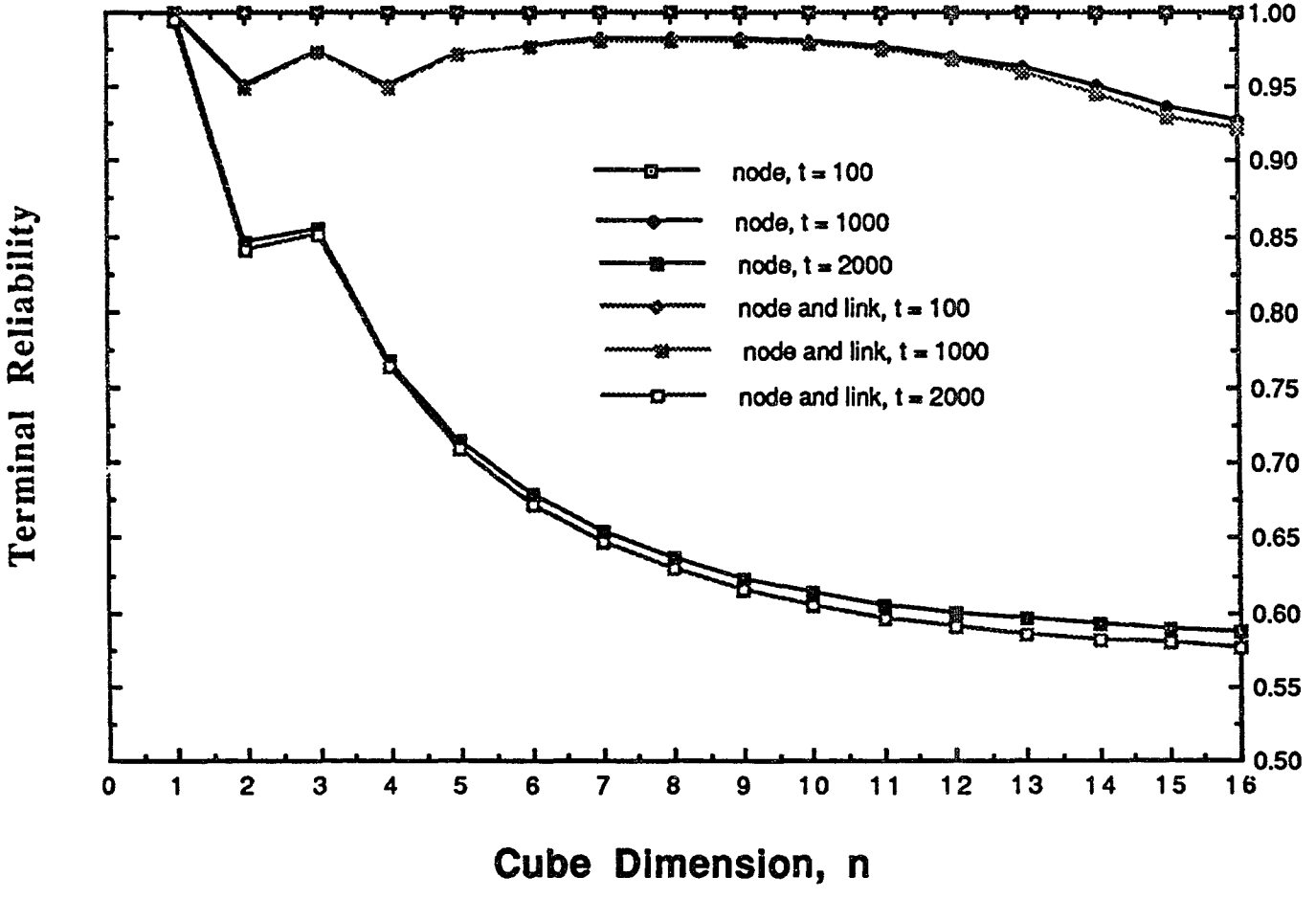


Figure 4.6 2CF Measure vs. Dimension for n-cube

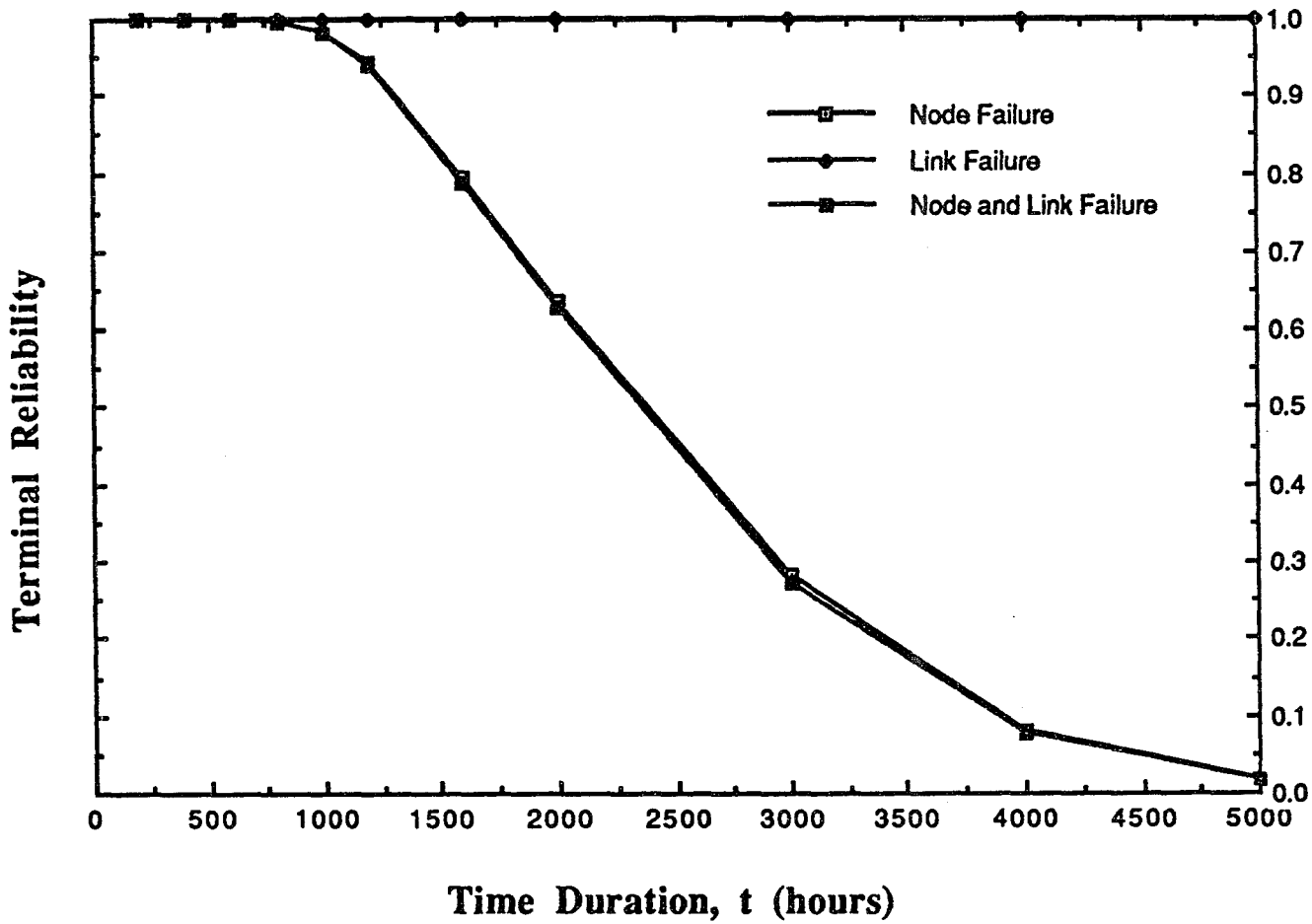


Figure 4.7 2CF Measure vs. Time for an 8-cube

4.5. Bounds on TBF Measure

4.5.1. Using a Markov Model

Najjar and Gaudiot [54,55] have proposed a method to approximate the task-based functionality (TBF) measure for an n-cube with failing nodes. Their method uses disconnection probability. A system is said to be disconnected if its graph topology has two or more disconnected components. A subset of k connected nodes is called a cluster, and a cluster is disconnected from the system if and only if all its neighbors have failed. Let

$Q^N(i)$ = **Pr**{disconnection occurs at the ith PE failure | no disconnection occured at the $(i-1)$st failure} and

${Q_k}^N(i)$ = **Pr**{disconnection of a k PE cluster occurs at the ith PE failure | no disconnection occured at the $(i-1)$st failure}.

The total disconnection probability is given by

$$Q^N(i) = \sum_{k=0}^{N-1} {Q_k}^N(i). \tag{4.40}$$

Computing Equation (4.40) for all values of k is practically intractable and, thus, the reference [54] assumes that k is a power of 2. Furthermore, by using Monte-Carlo simulation, the authors [54,55] show that the probability of disconnecting a single PE, ${Q_1}^N(i)$ is the dominant component of $Q^N(i)$ and thus suggest computing only ${Q_1}^N(i)$ for the $Q^N(i)$. The conditional probability of disconnecting a single PE in the n-cube is given as [54]:

$$Q_1{}^N(i) = \begin{cases} 0 \quad ; \text{for } i < n \\ \dfrac{N}{\binom{N}{n}} \quad ; \text{for } i = n \\ \dfrac{nN\binom{N-n-1}{i-n}}{(N-i+1)\binom{N}{i-1}} \quad ; \text{for } i > n \end{cases} \tag{4.41}$$

The TBF measure is, then, computed by using a general Markov model of failures shown in Figure 4.8 with $D \leq \frac{N}{2}$. Given a node failure rate γ, the rates of state transitions, η_i and μ_i, are expressed as a function of the single node failure rate as follows:

$$\eta_i = C_i(N - i)\gamma\,; \quad \mu_i = (1 - C_i)(N - i)\gamma.$$

Here, C_i is a state-dependent coverage factor, which is given as

$$C_i = c \cdot (1 - Q^N(i)),$$

where c is the coverage factor for the recovery scheme. The probability of being in state i, $P_i(t)$ is obtained as [54]:

$$P_i(t) = \left[\prod_{j=0}^{i-1} C_j\right]\binom{N}{N-i}(e^{-\gamma t})\,(1 - e^{-\gamma t})^i, \tag{4.42}$$

where i is the number of failed nodes at time t. In the Markov model, the TBF metric, $TBF(n, K, r)$, for n-cube with node reliability $r = e^{-\gamma t}$ and requiring at least K healthy nodes to be connected is simply the probability of being in one of the states 0 through D, for $K = N-D$. This is given as

$$TBF(n, K, r) = \sum_{i=0}^{D} P_i(t). \tag{4.43}$$

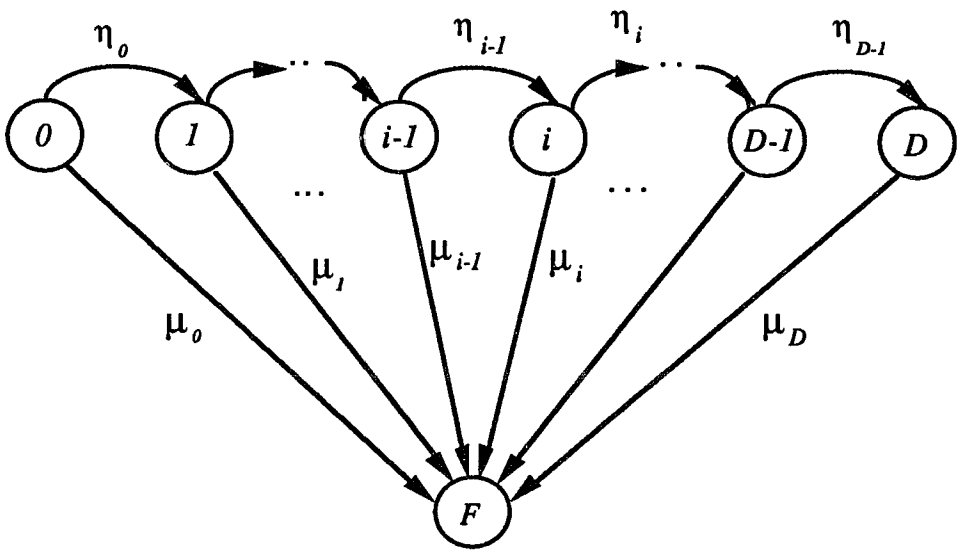


Figure 4.8. Markov Model for TBF Bounds

Equation (4.42) is computable in a time or space polynomial in the order of i. Since the variable i used in (4.42) can potentially go up to $N-1$, the time or space complexity involved in the equation is exponential in the order of the dimension of the cube. In what follows, we describe an alternative polynomial time algorithm utilizing the disconnection probabilities given by Equation (4.40).

4.5.2 Proposed Technique for TBF Bounds

In this section, we present a new algorithm to obtain the upper bound on the TBF measure for a hypercube. For a C_n with node reliability r, let us define

$$R(I, r) = \mathbf{Pr}\{\text{Exactly } I \text{ healthy nodes are connected}\}, \tag{4.44}$$

and thus,

$$TBF(n, K, r) = \sum_{I=K}^{N} R(I, r). \tag{4.45}$$

We divide the problem of computing Equation (4.45) into two cases: C1 and C2. Case C1 computes exact $R(I, r)$ for $I \geq N-n$. On the other hand, Case C2 obtains bounds on the $R(I, r)$ for $I < N-n$. In the following we present the computation of the two cases.

Case C1: For $N - n \leq I \leq N$.

$$R(I, r) = \begin{cases} \binom{N}{I} r^I (1-r)^{N-I} & ; \text{for } I > N - n \ , \\ \left(\binom{N}{I} - N\right) r^I (1-r)^{N-I} & ; \text{for } I = N - n \end{cases} \tag{4.46}$$

Note, Equation (4.46) obtains the exact reliability for $R(I, r)$. When there are less than $(n-1)$ node failures, the remaining operational nodes are still connected since the node connectivity of a C_n is n. When exactly n nodes fail, all possible combinations of the remaining nodes are connected except the N combinations when the n failing nodes are the neighbors of a particular node.

Case C2: For $I < N-n$.

This case computes the bounds on the $R(I, r)$ based on the disconnection probability (discussed in Section 4.5.1). To obtain an upper bound on the $R(I, r)$, we first consider Equation (4.44) as

$$R(I, r)=\mathbf{Pr}\{\text{Exactly } I \text{ nodes are healthy}\}\cdot\mathbf{Pr}\{\text{The } I \text{ nodes are connected}\} \quad (4.47)$$

Then, the second term of Equation (4.47) is computed using the disconnection probability $Q^N(i)$ as

$$\mathbf{Pr}\{\text{The } I \text{ nodes are connected}\} = (1 - Q^N(N-I)). \quad (4.48)$$

Notice that $Q^N(N-I)$ represents the probability that disconnection occurs at the $(N-I)$th node failure given no disconnection occured at the $(N-I-1)$th failure. Thus, $(1 - Q^N(N-I))$ is the probability that there is no disconnection when $(N-I)$ nodes fail. Following Najjar and Gaudiot's suggestion [54,55], we compute the $Q^N(i)$ using Equation (4.41), and thus,

$$R(I, r) = \begin{bmatrix} N \\ I \end{bmatrix} r^I(1-r)^{N-I} \cdot (1 - Q_1{}^N(N-I)). \quad (4.49)$$

Lemma 4.3. Equation (4.45) obtains an upper bound on the TBF measure.

Proof: Equation (4.47) computes the upper bound on the $R(I\ r)$ since the events {Exactly I nodes are healthy} and {The I nodes are connected} are considered to be independent. Furthermore, we have used ${Q_1}^N(i)$ for $Q^N(i)$, where the former always has a smaller value than the latter. □

4.5.3 Discussion

Table 4.6 provides experimental results of the bounds on the TBF measure for some hypercubes with different system life times. The table compares the upper bound results (UB) obtained by our new method with the approximate values computed by the techniques given by Najjar-Gauidiot (NG) [54,55] and Kim *et al.* [42]. Note, we are not able to write a program for the technique given in [42] since the reference does not provide complete steps. Thus, for Kim *et al.*'s technique we have used the results provided in reference [42]. We consider only for $K = \frac{N}{2}$ and $\frac{N}{4}$. The results show that the Kim *et al.* method [42] always gives approximation values which are higher than the upper bounds on the reliability. Table 4.7, on the other hand, shows that the results obtained by Najjar-Gaudiot [54,55] fluctuate above or below the upper bound results.

Table 4.6
Bounds on TBF Measure for C_7
(γ = 2,500 FITS)

time (*h*)	K = 64			K = 32		
	NG	Kim *et al.*	UB	NG	Kim *et al.*	UB
400	1.0	1.0	1.0	1.0	1.0	1.0
800	0.9995	1.0	0.9999	0.9995	1.0	0.9998
1200	0.9937	1.0	0.9986	0.9937	1.0	0.9986
1600	0.9679	1.0	0.9945	0.9679	1.0	0.9945
2000	0.8973	0.9943	0.9799	0.9005	1.0	0.9852
2400	0.7160	0.8844	0.8610	0.7773	1.0	0.9691
2800	0.3733	0.5043	0.4865	0.6081	1.0	0.9456
3200	0.1010	0.1438	0.1380	0.4254	1.0	0.9149
3600	0.0131	0.0020	0.0194	0.2654	0.9997	0.8787
4000	0.0010	0.0015	0.0015	0.1483	0.9954	0.8378

Table 4.7
Bounds on TBF Measure for C_n, and $K = \frac{N}{2}$

C_n	γt	NG	UB
4	0.1	0.9998	0.9990
	0.2	0.9949	0.9896
	0.3	0.9717	0.9607
	0.4	0.9124	0.7964
	0.5	0.8086	0.7964
5	0.1	0.9999	0.9998
	0.2	0.9975	0.9972
	0.3	0.9830	0.9870
	0.4	0.9376	0.9562
	0.5	0.8344	0.8736
10	0.3	0.9990	0.9999
	0.4	0.9895	0.9997
	0.5	0.9447	0.9986
	0.7	0.2786	0.4223
	0.8	0.0004	0.0006
16	0.4	0.9991	0.9999
	0.5	0.9865	0.9999
	0.6	0.8948	0.9994
	0.7	0.0242	0.0405

Chapter 5

Deterministic Fault Tolerant Measures

The deterministic model considers that system component(s) has (have) failed, and aims to determine the hypercube functionality using criteria such as NCF, TBF, SF, etc. Fault tolerant broadcasting in a hypercube is an important topic that belongs to the deterministic NCF model. The system is said to be functional if all working PE's are connected. The need for broadcasting appears in distributed agreement and clock synchronization [67]. Furthermore, a distributed hypercube reconfiguration method [48] assumes that each PE has component failure information which is to be broadcast by the host/manager of the system. In the presence of faulty components, broadcasting is not straightforward since a faulty PE may omit, corrupt, reroute or alter the information passing through it. This dissertation provides an overview of literature on this topic. We present a hybrid approach to fault-tolerant broadcasting that uses the concepts of redundant and non-redundant methods. An issue related to reliable broadcasting is fault tolerant routing in which the system is considered to be operational if there exists at least one path between two given PE's. The reliable routing problem belongs to the deterministic 2CF model. This topic has been addressed in reference [50] and will not be discussed in this dissertation.

Esfahanian [27] has shown that a hypercube C_n can tolerate up to $2n-3$ node failures and can remain connected provided that the failed nodes do not isolate any

operational PE in the system. In other words, there is no C_0 disconnection. His model can be considered a restricted class of the deterministic TBF criterion. For this model, it is assumed that either nodes or links are reliable and that a forbidden fault set exists. In [27], a forbidden fault set is defined as the faults that cause disconnection of a working 0-subcube, C_0. This class of TBF model is particularly applicable to the large scale degradable hypercubes used to run concurrent algorithms requiring at least K connected processors. It is further assumed that the running algorithms are not sensitive to changes in the system topology. In this dissertation, we show that a C_n can tolerate up to $3n-6$ faulty PE's and remain connected if neither C_1 nor its subsets are disconnected. This assumption is not impractical as researchers have studied the probabilities of C_i disconnection and have shown that for C_1 the disconnection probability is very low [27,54].

Many parallel algorithms designed for a C_n can also be executed on a C_k ($k \leq n$) with a slow-down factor of 2^{n-k} [57]. When system components start failing, a degradable hypercube used for this class of programs is reconfigured to a smaller sized subcube. It is important, however, to obtain the largest operational subcube so that the degraded hypercube operates with minimal performance reduction. Several methods to identify the largest operational subcube (LOS) have recently been described in the literature [9,18,25,48,56,65,79]. These methods are classified into centralized and distributed categories. The centralized approach uses a processor (usually the supervisory PE of the system) to generate the LOS. The distributed approach, on the other hand, allows each processor v to identify the LOS which includes the processor v. The dissertation discusses the merits and demerits of these approaches. In addition, we propose efficient distributed algorithms for the LOS identification problem. Our method uses the CMB operator of CAREL which includes the multiple variable inversion concept. In case the number of non

available nodes (faulty or busy) increases, an alternative distributed approach processes m available nodes in $O(mn)$ time to solve the LOS problem in C_n. Note, the LOS identification problem belongs to the deterministic model with SF measure.

5.1 Deterministic NCF Model

Broadcasting is a process of transmitting information from a source PE to all other PE's. Efficient broadcasting is required for the hypercube so that good performance is achieved with the variety of algorithms including matrix multiplications, LU factorization, and database queries [41]. Most algorithms discussed in the literature [41,80] are applicable for the hypercube with non-faulty components. Fault tolerant broadcasting considers the faulty components and is of utmost importance. In what follows, we discuss a two-step broadcasting algorithm given in reference [80] (for fault free hypercube system) because most fault tolerant broadcasting algorithms are based on this method.

In step 1 of this algorithm, the source sends the message and its *weight* to its n neighboring nodes. A message is given a weight i if it is transmitted through an ith neighbor. A PE, after receiving the message, checks the weight. If the weight is 0, the PE merely keeps the message. Otherwise, as a second step, the message is sent to all the $0th$, $1st$, $\cdots$, (weight$-1)th$ neighbors. These two steps construct a broadcasting tree in which the root is the source PE, and the children are the remaining PE's. The broadcasting algorithm is implemented as follows:

Step 1. **for** (i = 0 to n-1) **do**

send (message, i, i);

Step 2. **for** (i = 0 to weight-1) **do**

send (message, i, i);

The function **send** (message, i, j) transmits the message with weight i to the jth neighbor. To illustrate the algorithm, consider a C_3 with PE (010) as the source. Figure 5.1 shows the resulting tree where the link label refers to the message weight. It is obvious that the algorithm does the broadcast in $O(n)$ steps.

5.1.1. Fault Tolerant Broadcasting Algorithm - An Overview

Fault-tolerant broadcasting algorithms [17,28,50,67,87] are classified into two categories, namely, redundant and non-redundant types. Each of these methods contains two major aspects to consider: the message transmission and reception. Reliable broadcasting based on the redundant approach can tolerate faulty components by sending multiple copies of the message through n disjoint paths. Thus, each PE receives more than one copy of a message. Non-redundant fault tolerant broadcasting, on the other hand, avoids intermediate faulty components on the communication paths and makes each PE receive only one copy of the message. The performance of a fault tolerant broadcasting algorithm is rated by its capability to tolerate the number of faulty components and by its speed to broadcast the message in terms of the number of transmission steps. The steps taken, in turn, depend on the node model used in the system. In a multi-channel node model, a PE can simultaneously receive and transmit messages in all of its incoming and outgoing links. On the other hand, a single-channel PE model allows each PE to send the message through, at most, one outgoing link at a time.

5.1.1.1 Using a Redundant Approach

Fault tolerant broadcasting based on the redundant approach is achieved by sending multiple copies of the message through n disjoint paths [67]. This class of broadcasting is simple since it does not require backtracking during the broadcast.

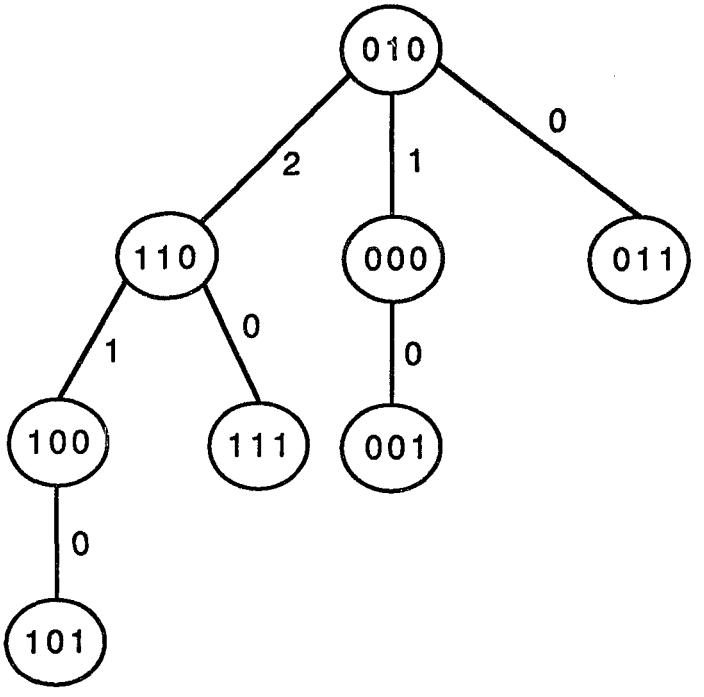


Figure 5.1. A Fault-Free Broadcasting Tree

The approach, to its advantage, does not need the faulty component information of the system. This advantage is especially important in critical real-time applications which cannot tolerate the time overhead to identify the faulty components. Message transmission in the redundant approach is trivial. The method requires the source PE to send the message to all its neighbors. Then, the neighbors use a coordinated recursive doubling to broadcast the message to all the PE's. The recursive doubling is coordinated to ensure that each node gets the broadcast message through disjoint paths. For this step, we provide algorithm Redundant1 which works the same as Algorithm A given in reference [67] presented more concisely. Redundant1 is based on the coordinated recursive doubling method and is given as follows.

Algorithm Redundant1

/* executed by the source PE */

for (*i=0* **to** *n-1*)

  send (*message, 0, i, i*);

for (l=x **to** n-1)

  send (*message*, l+1, y, (l+1+y) **mod** n);

The function **send** (message, x, y, z) denotes that a PE receiving a message with parameter x from *ancestor* y sends the message to its neighbor z. The parameter x lets a PE transmit the message to its $n-x$ neighbors, and the parameter y denotes that the ancestor of the path to the PE is the yth neighbor of the source PE. The redundant broadcasting algorithm guarantees correct broadcast provided there are fewer than n faulty components since, otherwise, the faults may isolate a PE in the C_n. Redundant1 completes the broadcast in n+1 steps for the multi-channel PE model. The result is obvious since the diameter of a C_n is n, and, hence, any PE is at most n steps away from any other PE. The last step is taken by the last receiving PE to complete the message reception. Reference [28] proves that when using a

multi-channel PE model with, at most, $n-1$ faulty PE's, the broadcasting requires $n + O(n)$ steps, irrespective of the component fault models. However, for a single-channel PE model, Redundant1 takes a total of $2n$ steps. The sender needs n steps to send the message to its n neighbors, which start their coordinated recursive doubling right after they receive the message. The last n steps are taken by the last neighbor to receive the message for its recursive doubling. Table 5.1 shows the disjoint paths to each PE for a C_3 with PE (010) as the source, where the number in a parentheses represents the parameter x of the received message. As the result, each PE receives more than one copy of the message. The receiving PE, thus, must be able to identify the correct data from the multiple copies of the message it has received since faulty components may omit, corrupt, reroute, or alter the message passing through them. When the hypercube is fault-free, each PE gets n identical copies of the message.

Reference [67] has discussed three component fault models, namely simple omission faults, non-Byzantine faults, and Byzantine faults. In a simple omission fault model, a faulty component either sends the message correctly or does not send any message at all. Since there is no corrupted data, a receiver does not identify the correctness of the message and, hence, Redundant1 tolerates up to $n-1$ faulty components. The non-Byzantine model considers a faulty component able to corrupt the message passing through it. Reference [67] suggests using majority voting to identify the correct data from the multiple copies of message. Using this approach, Redundant1 tolerates up to $\lfloor n/2 \rfloor$ faults. The third fault model, Byzantine fault, considers a faulty PE able to omit, corrupt, reroute, and even lie [46]. For this fault model, one uses majority voting with at least $\lceil 2n/3 \rceil$ copies of received message

Table 5.1.
Redundant Broadcasting in C_3, Source (010)

PE	Paths via the bth neighbor		
	b=0, PE 3	b=1, PE 0	b=2, PE 6
0	2-3-1 (3)	2 (0)	2-6-4 (3)
1	2-3 (1)	2-0 (2)	2-6-7-5
3	2 (0)	2-0-1 (3)	2-6-7 (3)
4	2-3-1-5 (3)	2-0 (1)	2-6 (2)
5	2-3-1 (2)	2-0-4 (2)	2-6-7 (2)
6	2-3-7 (3)	2-0-4 (3)	2 (0)
7	2-3 (2)	2-0-4-5 (3)	2-6 (1)

as the quorum, and, thus, the Redundant1 tolerates up to $\lfloor n/3 \rfloor$ faults. However, if the message is authenticated by an unforgeable digital signature, Redundant1 tolerates up to $n-1$ faults irrespective of the fault models [67].

5.1.1.2 Using a Non-Redundant Approach

Non-redundant fault tolerant broadcasting algorithms, described in the literature [5,17,87], rely on a construction of a spanning broadcasting tree. For that, the methods need local (global) fault information, in which each PE has a list of faulty neighbors (components in the system). Non-redundant broadcasting achieves fault tolerance by avoiding intermediate faulty components on the communication paths. As a result, each PE receives only one copy of the broadcast message, and, hence, the receiving nodes do not need any message identification mechanism like that used in the redundant approach. To construct a spanning broadcasting tree, Al-Dhelaan and Bose [5] modify the two-step algorithm given in [80]. Their method assumes that each PE has local fault information. Each message is given a weight such as that in [80]. However, when a PE identifies a faulty neighbor, the node sends the message, its weight, and the weight of the message that was supposedly for the faulty PE to another healthy neighbor. However, the algorithm is applicable only if there is one faulty PE, or if each PE is connected to at most one faulty PE. Thus, a better and more general approach for the broadcasting problem is needed.

Lee and Hayes [50] define an operational PE as unsafe if it has at least two unsafe or faulty neighboring nodes. Thus, a PE can be in one of three states: faulty, healthy, or unsafe. Furthermore, the authors define a C_n unsafe if the cube contains only unsafe or faulty PE's. Figure 5.2 illustrates a C_3 with faulty PE's (101) and (110) and with the unsafe PE's. The notion of an unsafe state is important since it may lead to backtracking due to its proximity to faulty components. Based on this

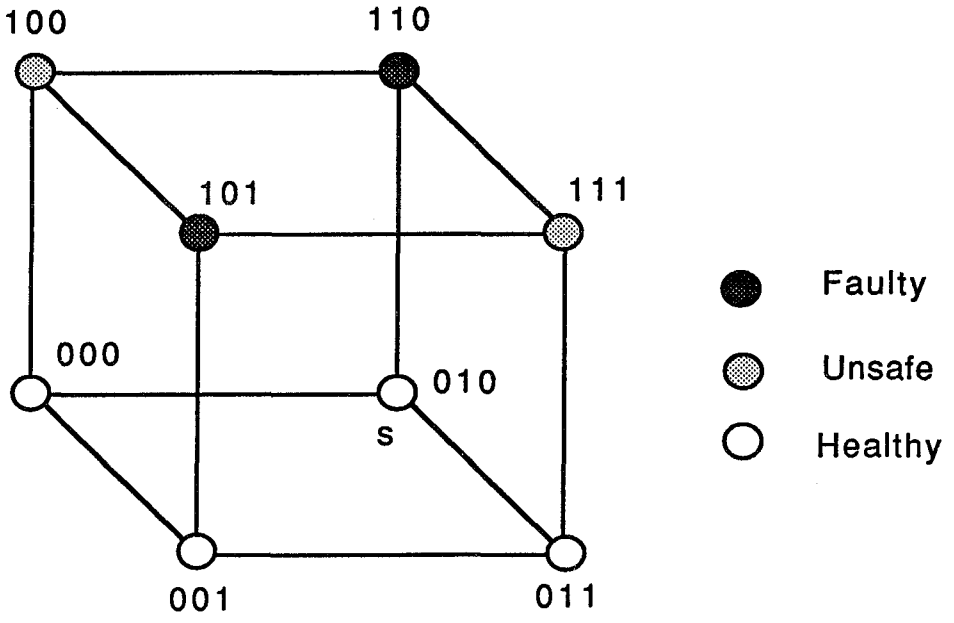


Figure 5.2. An Illustration for Unsafe PE

concept, the authors have proposed an algorithm which can tolerate up to $\lceil n/2 \rceil$ PE failures [50]. The algorithm assumes that each PE has local fault information and uses Find_unsafe routine to obtain the unsafe states. This step is completed in $O(f^3)$ steps, where f is the number of faults. Then, from the unsafe PE information, a modified version of the broadcast algorithm given in reference [80] is used to generate a broadcasting tree. The modified method positions the unsafe PE's as the last PE's to receive the message, and, therefore, no unsafe PE's must broadcast the message. Figure 5.3 shows the generated broadcasting tree for the C_3 given in Figure 5.2. Note, the link label denotes the step number of the message transmission. Provided that C_n is not an unsafe cube, the algorithm [50] runs in n steps in either single-channel or multi-channel PE models. It is shown in [50] that the least number of faults needed to make a C_n unsafe is $\lceil n/2 + 1 \rceil$, and, hence, the given algorithm can tolerate up to $\lceil n/2 \rceil$ faults. When there is no PE failures, this algorithm reduces to the algorithm given in reference [80]. Wu and Fernandez [87] have improved Lee and Hayes' algorithm to obtain a better method that can tolerate more PE failures. An unsafe PE is defined as a healthy PE with the following two properties [87]:

- There are at least two faulty neighbors, and
- There are at least three unsafe or faulty neighbors.

To generate the unsafe PE information, reference [87] uses a Global_status routine. Then, the modified unsafe PE information, in turn, is utilized in a fault tolerant broadcasting algorithm. Wu and Fernandez [87] conjecture that there is no assignment of $n-1$ faulty PE's that makes a C_n unsafe and then conclude that their algorithm can tolerate up to $n-1$ faults.

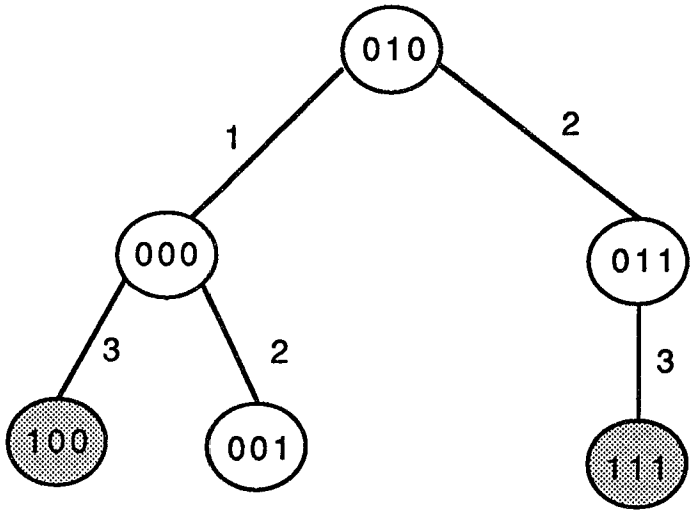


Figure 5.3. Broadcasting Tree with Unsafe PE Information

5.1.2 Hybrid Technique

The redundant broadcasting algorithm is attractive for its simplicity. Furthermore, the algorithms based on this approach do not need fault information. The message transmission is simple since each PE merely sends the message through its outgoing CE's. Since each PE receives multiple copies of message, however, the receiving PE's must be able to identify the correct message. This requirement limits the fault tolerant capability of such approach to up to $\lfloor n/2 \rfloor$ and $\lfloor n/3 \rfloor$ faulty components for non-Byzantine and Byzantine fault models, respectively. The non-redundant broadcasting algorithms, on the other hand, require that each PE has at least limited fault information. The methods based on this approach start by constructing a broadcasting tree which requires the unsafe PE information. Obtaining the unsafe PE information is not a trivial task and requires $O(n^3)$ steps. However, once the broadcasting tree is available, the message transmission and reception are easy. The redundant broadcast outperforms the non-redundant method, especially for multi-channel PE model with simple omission faults. The former can tolerate up to $n-1$ faults (for the simple omission model), while the latter is conjectured (in reference [87]) to be able to tolerate the same number of faults. Furthermore, the redundant approach needs $O(n)$ steps compared to $O(n^3)$ steps required in the non-redundant method (including the algorithm to obtain unsafe PE information).

In this section, we propose a hybrid technique, Redundant2, which utilizes the advantages of the redundant and non-redundant approaches. On the one hand, Redundant2 uses the simple transmission step of the redundant approach and takes advantage of the $O(n)$ broadcasting speed of Redundant1. On the other hand, the hybrid approach implements the feature used in the non-redundant approach, namely avoiding the faulty PE's in the communication paths to improve the fault tolerant capability of the approach. In Redundant2, each PE sends the message only

to its healthy neighbors, and, thus, only correct copies of messages are broadcast in the cube. Consequently, each node will receive only the set of correct messages, which makes the message identification mechanism used in [67] unnecessary. This step makes Redundant2 tolerate up to $n-1$ faults, irrespective of the component fault models. Furthermore, the message reception mechanism in each receiving PE is modified to recognize only the first arriving copy of the message and to ignore the rest of the copies. This technique further reduces the computer time for message reception (compared to that in Redundant1). Redundant2, however, requires each PE to have local fault information (like in non-redundant approach). This requirement, nevertheless, is not hard to implement since following the test method given in reference [7], each PE in the cube can test its neighbors simultaneously. In what follows, we provide Redundant2.

Algorithm Redundant2 /* Proposed Hybrid Approach */

/* executed by the source PE s */

for (i=0 **to** n-1)

  if (the ith neighbor is healthy)

    send (message, 0, i, i);

/* executed by each PE receiving message with parameter x and ancestor y */

for (l = x **to** n-1)

  if (the {(l+1+y) **mod** n)}th neighbor is healthy)

    send (message, l+1, y, (l+1+y) **mod** n);

Table 5.2 provides the result obtained by algorithm Redundant2 for a C_3 with PE (010) as the source, and PE's (101) and (110) as the faulty components.

Table 5.2.
Hybrid Broadcasting in C_3, Source (010), and Faults (101) and (110)

PE	Paths via the bth neighbor		
	b=0, PE 3	b=1, PE 0	b=2, PE 6
0	2-3-1 (3)	2 (0)	-
1	2-3 (1)	2-0 (2)	-
3	2 (0)	2-0-1 (3)	-
4	-	2-0 (1)	-
5	-	-	-
6	-	-	-
7	2-3 (2)	-	-

5.2 Deterministic TBF Model

Researchers have mainly used graph theoretic concepts, particularly the notion of connectivity, to develop the deterministic fault tolerant model. However, if the connectivity concept is applied as such, the measure lacks generality since it does not take into account the different types of disconnected hypercubes. This conventional connectivity concept considers any disconnectivity as the result of removing n neighbors from a certain PE. It implies that the parameter does not consider the severity of the damage to the system caused by different PE failures, which makes the measure inaccurate for certain applications. To remedy this deficiency, Esfahanian [27] has suggested using generalized measures of *connectedness*, such as the notions of toughness and mean connectivity of a graph. The conventional connectivity concept assumes that any subset of nodes can potentially fail at the same time. However, when PE's fail independently but with different probabilities, equal-size faulty sets may have different probability of failures. Esfahanian [27] defines a set of components which do not fail at the same time as a forbidden faulty set. One may consider a set of components with negligible probability of failures as forbidden faulty sets to find out what it takes to disrupt the functionality of the system. The result leads to obtaining the upper bound on the reliability of the system. Reference [27] has used any node's n neighbors as the forbidden faulty sets. Note, for a C_n, the total number of such sets is very small compared to all other possible equal-size sets, i.e., 2^n to $\begin{bmatrix} 2^n \\ n \end{bmatrix}$. Thus, the author suggests ignoring these forbidden faulty sets to obtain a reliability measure "beyond" that obtained by connectivity concept. Theorem 5.1 shows the result. To avoid trivial cases, assume that $n \geq 4$.

Theorem 5.1 [27]. A C_n can tolerate up to $(2n - 3)$ PE failures and remains connected provided that the failures do not disconnect any subcube C_0.

Proof: Refer to [27] □

In what follows, we generalize Theorem 5.1 by extending the forbidden faulty sets. Here, we consider the neighbors of a C_1 belonging to the forbidden set. The result is given in Theorem 5.2. For the following discussion, we have used some notations given in [27]. In particular, let $V(G)$ denote the set of nodes in G, and $A(G:v)$ represent a set of all nodes which are adjacent to node v in G. When network G is known from the context, $A(G:v)$ will be denoted by $A(v)$.

Theorem 5.2. A C_n can tolerate up to $(3n - 6)$ PE failures and remain connected provided that the failures do not disconnect any C_1 or its subsets.

Proof: Let F be the set of nodes which do not disconnect any C_1 or its subsets in C_n. Consider F as an arbitrary subset of $V(C_n)$ such that $|F| = 3n-6$ and for any two connected nodes $u,v \notin F$, either $\{A(u) - v\} \not\subset F$ or $\{A(v) - u\} \not\subset F$. By the definition of the set F, there must be a node w that is connected to u or v. Without loss of generality, consider w be connected to the node u, and let us define

$$S = \{A(u) - v - w\} \cup \{A(v) - u\} \cup \{A(w) - u\}.$$

Observe that $|S| = (n-2) + (n-1) + (n-2) = 3n-5$, and that network $C_n - S$ is disconnected. We now complete the proof by showing that $C_n - F$ is connected.

Let us consider the C_n as composed of two *congruent* $(n-1)$-cubes such that each node on one $(n-1)$-cube is connected to its congruent node on the other $(n-1)$-cube. Let L denote the first $(n-1)$-cube, and R represent the other $(n-1)$-cube. All 2^{n-1} links connecting L and R are termed as *external* links, and the links within L and R are called *internal* links. Define $F_L = V(L) \cap F$ and $F_R = V(R) \cap F$. The size of set F and the fact that $F_L \cap F_R = \varnothing$ imply that either $|F_L| \leq \left\lfloor \frac{3n-6}{2} \right\rfloor$ or $|F_R| \leq \left\lfloor \frac{3n-6}{2} \right\rfloor$. Without loss of generality, consider

$|F_R| \leq \left\lfloor \frac{3n-6}{2} \right\rfloor$. It means that network $R - F_R$ is connected since R is a C_{n-1}, and by Theorem 5.1 the network tolerates up to $2(n-1)-3 = 2n-5$ node failures. Note, $2n-5 \geq \left\lfloor \frac{3n-6}{2} \right\rfloor$ for $n \geq 3$. It remains to be shown that any node in network $L-F_L$ is connected via a path to a node in network $R-F_R$. Let u_L be an arbitrary node in $L-F_L$ and let (u_L, u_R) be its corresponding external link. If $u_R \notin F_R$ we are done. So assume that $u_R \in F_R$. Since $A(u_L) \not\subset F$, there exists a vertex w_L in $L-F_L$ which is connected to u_L. Let (w_L, w_R) be the corresponding external link connecting node w_L with a node w_R in $R-F_R$. Again, if $w_R \notin F_R$ we are done. Therefore, assume $w_R \in F_R$. This and the fact that $\{A(u_L)-w_L\} \not\subset F$ or $\{A(w_L)-u_L\} \not\subset F$ imply that there is a node x_L in $L-F_L$ that is connected to either u_L or w_L. Without loss of generality, consider x_L be connected to w_L, and let

$$X = \{A(L:u_L)-w_L\} \cup \{A(L:w_L)-u_L\} \cup \{A(L:x_L)-w_L\},$$

and $F' = F - \{u_R, w_R\}$. Note that $|X| = 3n-7$ since L is an $(n-1)$-cube. For each node $y_i \in X$, $1 \leq i \leq 3n-7$, let (y_i, z_i) be its corresponding external link. This means that $z_i \in V(R)$. Since $|F'| = 3n-8$ and each node in F' can correspond to at most one such external link, there must exist an external link (y_i, z_i) for some i such that nodes $y_i, z_i \notin F'$ (and consequently $y_i, z_i \notin F$). This implies that in network $C_n - F$, node u_L is connected to a node $z_i \in V(R-F_R)$ via a path of length at most 4. Figure 5.4 illustrates the proof. □

Table 5.3 shows the comparisons of the sturdiness parameters of the C_n computed by the conventional PE connectivity concept, the Esfahanian concept and our proposed forbidden faulty PE's for $n = 4, \cdots, 16$.

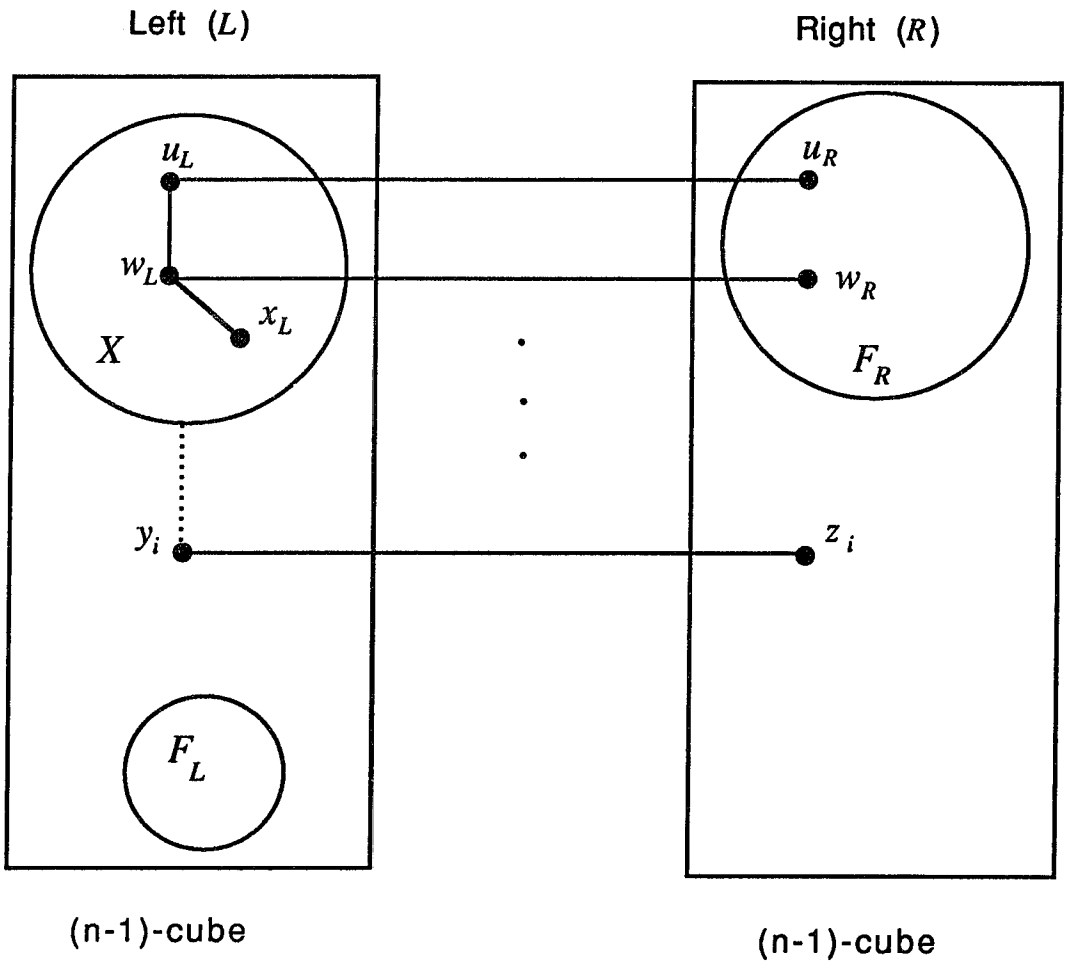


Figure 5.4. Illustration for the Proof of Theorem 5.2

Table 5.3.
A Deterministic TBF Measure for Hypercube

C_n	The Total Number of Tolerable Faults		
	Conventional	**Theorem 5.1**	**Theorem 5.2**
4	3	5	6
5	4	7	9
6	5	9	12
7	6	11	15
8	7	13	18
9	8	15	21
10	9	17	24
11	10	19	27
12	11	21	30
13	12	23	33
14	13	25	36
15	14	27	39
16	15	29	42

5.3 Deterministic SF Measure

5.3.1 Background

The performance of a hypercube system in a multiuser-multitasking environment depends on efficient subcube allocation for the incoming jobs. In a multiuser environment, some processing elements may have already been allocated for other tasks and therefore are unavailable for the new jobs. A subcube allocation method is used to obtain a cube of the requested size from the remaining available nodes. The method must fail only when the requested cube size does not exist. References [6,16,24,43] discuss some subcube allocation methods. The methods assume that the system components are reliable. In reality, however, the components are not fully reliable, and, therefore, the subcube allocation methods, for this case, are not applicable.

A related and equally important issue is that of the largest operational subcube (LOS) identification. This topic considers the failed system components and aims to obtain the largest subcube from the remaining operational components. Obtaining the largest subcube is advantageous so that the hypercube can run in a degraded mode with a minimal performance reduction. Many parallel algorithms designed to run on an n-cube also run on a k-cube ($k \leq n$) with a slowdown factor of 2^{n-k} [57]. The LOS identification problem belongs to the deterministic SF measure. Information on either faulty or operational components is assumed to be available.

Algorithms for diagnosing faulty nodes and links have been addressed in references [7,12,23]. The set of faulty nodes and links can be identified by the operational processors. Reference [23] provides a distributed fault diagnosis algorithm for total number of faulty components $r \leq n$. Once the faulty elements are identified, graceful degradation is achieved by reconfiguring the operational components and

the parallel algorithms in the system. Most algorithms designed for hypercube are adaptable to cube dimension changes. The algorithms usually pass the dimension of the host hypercube as their parameters, and hence, reconfiguring parallel algorithms is trivial. Thus, the reconfiguration problem is reduced to finding the LOS.

Algorithms for LOS identification have recently been discussed in the literature. The algorithms can be classified into two categories: centralized and distributed. The centralized approach uses a processor, like the supervisory or host of the cube, to run a LOS identification technique. It is assumed that the host has information on faulty or operational components. The approach generates a maximum sized subcube which we call as a *global* LOS. In a distributed approach, a node v executes a program to identify a κ-subcube that includes the node v. We call the subcube a κ-cube of node v ($SC_\kappa(v)$) or *local* LOS. Note, a local LOS is not necessarily as large as the global LOS. However, the maximum sized local LOS is a global LOS. For some cases, a distributed approach is preferable since it relieves the host from having to communicate with all nodes to identify the LOS. The distributed approach assumes that each node has a list of faulty or operational components. If only the supervisory node has the information, it is broadcast using any reliable broadcast algorithms. Most distributed methods assume that there are less than n node failures in a faulty n-cube. Since reliable broadcasting algorithms for hypercubes tolerate up to $n-1$ component failures (refer to Section 5.1), the assumption is justified.

References [25,56,65,66] discuss centralized subcube identification algorithms for the n-cube in the presence of node and/or link failures. Ozguner and Aykanat [56] utilized the inclusion-exclusion principle [20] to obtain the cube dimension, k, and the total number of available global LOS. Their method generates the k-subcubes from the faulty component information. The algorithm, first, encodes each

faulty element by extracting $(n-k)$ bits of its address at a particular $(n-k)$ coordinate positions. The $(n-k)$ bits, in turn, are used to compute $2^{\{0,1\}^{n-k}}$, where $\{0,1\}^{n-k}$ denotes the $(n-k)$-bit binary number. Then, the method uses the logic *or* operation on the encoded 2^{n-k} tuples. If the integer value of the resultant is $2^{2^{(n-k)}}$, the encoding from the chosen $(n-k)$ positions fails to obtain the LOS, and the method has to select other combinations of the $(n-k)$ coordinate positions. When a suitable set of $(n-k)$ coordinate positions is found, the algorithm decodes it to obtain a k-subcube. The technique in [56] always identifies a global LOS. For an n-cube with r faulty components, the method takes $O(2^r \log_2 r)$ steps to find the largest dimension k and $O(rk\begin{bmatrix} n \\ k \end{bmatrix})$ steps for the encoding-decoding phases [56]. Thus, the algorithm is not efficient for use in large hypercubes. Rai and Trahan [65,66] propose an algebraic technique which has improved time complexity over that of the method in [56]. Nevertheless, their algorithm takes $O(n^r)$ time. It is obvious that we need a better approach for the LOS identification problem.

Methods in [9,18,48] use faulty node information to generate the local LOS. Reference [48] considers the problem as a version of an NP-complete Hitting Set problem [29] and, hence, discusses a heuristic solution that generates a local LOS with high probability. For an n-cube with r faulty nodes, this heuristic algorithm takes $O(n^2 r^2)$ steps. The heuristic approach, however, requires exponential time to identify a global LOS since every healthy node has to carry out the algorithm, and $O(2^n - r)$ nodes are involved. Chen and Tzeng [18] use *reject* regions for their distributed method. A reject region is the smallest subcube that contains the faulty node and the antipodal of the local node. Each reject region is represented in a Boolean expression. First, their algorithm [18] obtains all reject regions, denoted in sum of Boolean terms, R. Then, it uses distributive law for Boolean expression [53]

on $P = \bar{R}$ to obtain the local LOS. In an n-cube with r faulty nodes, the algorithm [18] is empirically shown to have time complexity of $O(n^2r^2)$. The distributed algorithm can also be utilized to identify a global LOS. The authors [18] use only the healthy neighbors of each faulty node as local nodes. These selected nodes are referred to as *candidate* nodes. For an n-cube with r faulty nodes, there are $O(nr)$ candidate nodes, and, therefore, the total time complexity for identifying a global LOS is $O(n^3r^3)$.

In case the number of non-available nodes (faulty or occupied by some other tasks) increases, it is preferable to generate the LOS from the available (residual) nodes information. The residual nodes are represented either by specifying each of their addresses or by using a set of subcube descriptors. Reference [49] presents a centralized and a distributed method for this problem. However, both approaches require time exponential in the number of residual nodes. Note, Sridhar and Raghavendra [79] have shown that the LOS identification problem from the set of subcube descriptors is NP-hard. Therefore, the authors proposed a centralized technique which runs in polynomial time in the number of residual nodes.

5.3.2 Algebraic Technique for LOS Identification

In this section, we present an efficient distributed algebraic method for the LOS identification problem. Our proposed method obtains more than one possible local LOS from the faulty nodes information. The results from this method are beneficial because they allow the system manager to have more flexibility in reconfiguring the faulty system. The proposed algorithm can be used to identify global LOS. For an n-cube with r faulty nodes, we use $O(r)$ candidate nodes to

execute the algorithm and select the maximum sized local LOS as the global LOS. A modified CMB operator of CAREL (refer to Section 3.2) is used in the new approach.

5.3.2.1 Definitions and Notations

A subcube in a hypercube C_n is a subset of the n-cube that preserves the topological property of a hypercube structure. A k-subcube can be represented either by a list of its 2^k nodes or by its subcube descriptor. A subcube descriptor in a C_n is an n element string over symbols {0, 1, x}. The "0" and "1", and the "x" values in the descriptor are referred to as fixed or bound, and free coordinates, respectively. A k-subcube in a C_n has k free coordinates and $n-k$ fixed coordinates. Note, the total number of free coordinates refers to the subcube dimension. As an example, a subcube descriptor (xx101x) represents a 3-subcube of a C_6. The descriptor may also be denoted by the product of Boolean variables or product terms. A subcube descriptor is transformed into a product term by replacing a "1" ("0") in the ith position of the descriptor with an x_i ($\overline{x_i}$) and by ignoring the free coordinates. For example, the subcube (xx101x) is represented in a product term expression as $x_3\overline{x_2}x_1$. Notice that a C_n has null product term expression.

A k-cube of node v, $SC_k(v)$, is obtained by replacing any k bit positions of the address of v with free coordinates. In other words, the cubes are generated by expanding v towards its k (out of n) dimensional directions. Thus, each node v in C_n has $\begin{bmatrix} n \\ k \end{bmatrix}$ k-cubes. Let F_i represent a faulty node i and FD_i be a set of dimensions which provides a path from v to F_i. In other words, when node v is expanded towards all dimensional directions in FD_i, $SC_k(v)$ which contains F_i is obtained. Note, $k = | FD_i |$ for a k-cube, and the generated subcube that includes

at least one faulty node is referred to as a faulty subcube. On the other hand, a k-subcube that does not contain any faulty node is called an *operational subcube* (OS), and the operational subcube is a LOS if it is the largest possible OS. We use κ to denote dimension of the local LOS. The set FD_i is easily generated from the addresses of nodes v and F_i. A dimension j in FD_i refers to the jth position in the addresses of v and F_i when the two bits are different. An FD_i is generated by taking bitwise *xor* of the v address with F_i and considering a "1" in the jth bit position of the resultant as a dimension j in the FD_i. The FD_i's are partitioned into *independent* and *dependent* groups. An FD_i belongs to the independent group if $FD_i \cap FD_j = \varnothing$, for any j. Otherwise, it belongs to the dependent group.

Example 5.1. Consider Figure 5.5, where v = (001) and F_4 = (100). The bitwise *xor* of v with F_4 gives (101). The result is interpreted for FD_4 as {0,2}. If node v is expanded towards dimensional directions "0" and "2", we obtain an $SC_2(v)$ = (x0x) which includes F_4, and hence, is faulty.

In our proposed algorithm, the k-cubes of node v in C_n are represented in the Boolean expression as $\overline{X_1}\,\overline{X_2}\,\cdots\,\overline{X_{n-k}}$. For the new notation, each X_j denotes the product of Boolean variables $\{x_0, x_1, \cdots, x_{n-1}\}$, and for any pair of X_i and X_j, all variables in X_i are different than those in X_j. Here, we have used the multiple variable inversion (MVI) concept (discussed in Section 3.2) to represent the $SC_k(v)$'s in concise notations. Let $x_\alpha, x_\beta, \cdots, x_\eta$ be the variables in $X_1, X_2, \cdots, X_{n-k}$, respectively. For a local node v in C_n, $\overline{x_\alpha}\,\overline{x_\beta}\,\cdots\,\overline{x_\eta}$ represents an $SC_k(v)$ in C_n. For each $x_j \notin \{x_\alpha, x_\beta, \cdots, x_\eta\}$, a k-subcube is obtained by replacing each bit j in the address of node v with a free coordinate "x".

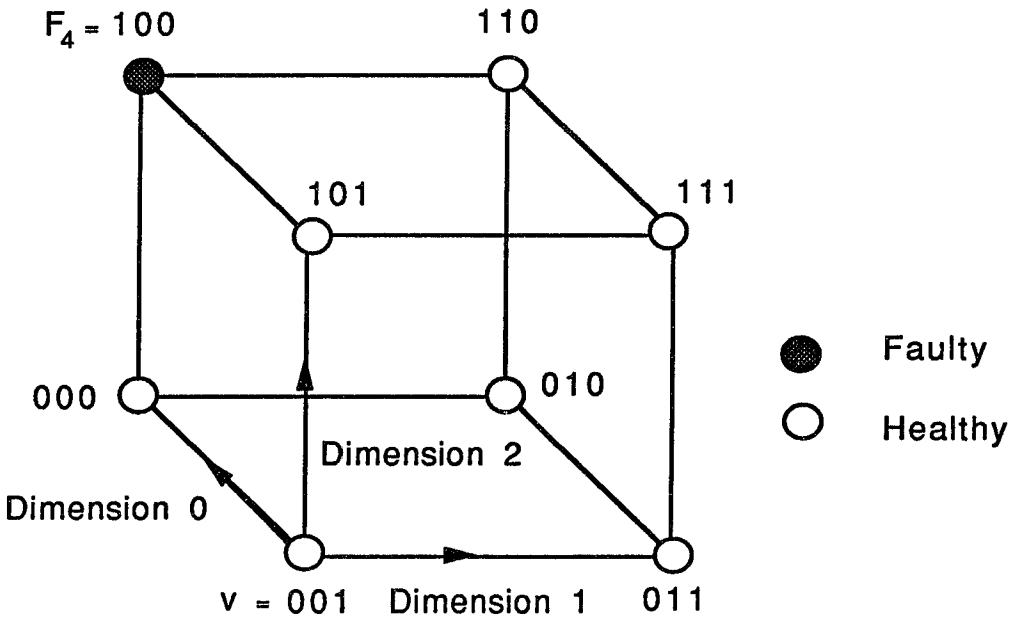


Figure 5.5. A 3-Cube of Node v

Example 5.2. For v = (0011), $\overline{X_1} = \overline{x_1 x_0}$ represents two 3-cubes: (xxx1) and (xx1x). The $\overline{x_1 x_0}$ denotes the events that node v is not expanded towards dimensional directions "0" or "1". On the other hand, $\overline{x_1 x_0}\,\overline{x_3 x_2}$ refers to four $SC_2(v)$'s: (x0x1), (0xx1), (x01x), and (0x1x).

5.3.2.2 Distributed Approach

Our proposed distributed method generates local LOS's by avoiding all possible faulty subcubes. For an FD_i, we use FB_i to represent products of Boolean variables. A Boolean variable x_j in FB_i represents a dimension j in FD_i. As an example, given FD_i = {0,2,4}, the corresponding FB_i is obtained as $(x_4 x_2 x_0)$. An FB_i is called an *independent* (*dependent*) *FB* if its corresponding FD_i belongs to independent (dependent) group. Note, an FB_i has logical value "1" or *true* if the local node v is expanded towards all dimensional directions in FD_i. Thus, the FB_i is *true* when a faulty $SC_k(v)$ is generated from v and the dimensions in FD_i. Obviously, non-faulty subcubes are generated when every FB_i is *false*. The result is stated formally in the following theorem.

Theorem 5.3. Given a set of faulty nodes F and a local node v, any generated $SC_k(v)$ does not contain any faulty nodes of set F if

$$\prod_{F_i \in F} \overline{FB_i} = 1. \tag{5.1}$$

Proof: By De Morgan's rule, Equation (5.1) is transformed into $\sum_{F_i \in F} FB_i = 0$. By definition, $FB_i = 0$ if v is not expanded towards all dimensional directions in FD_i, and, hence, the generated $SC_k(v)$ does not contain F_i. Since Equation (5.1) considers all F_i's, the theorem is proved. □

We use Equation (5.1) for our proposed distributed algorithm. First, the method generates an FD_i from a local node v and a faulty node F_i for all faulty nodes. Second, the technique removes any redundant FD_i and partitions the remaining FD_i's into independent and dependent groups. An FD_j is called *redundant* if an FD_i exists so that $FD_i \subseteq FD_j$. This step is used to reduce the overall time complexity of the algorithm. These two steps generate non-redundant independent and dependent FB_i's for use in Equation (5.1). Using Boolean identities, the equation can be solved to produce a sum-of-products expression [53] which represents the operational subcubes (OS) of node v. For our technique, we use a modified CMB operator (refer to Section 3.3.2), which we call CMB+ (discussed later). The CMB operator utilizes the multiple-variable inversion (MVI) concept (discussed in Section 3.2), and therefore generates the results for Equation (5.1) faster and more concisely. The operator generates the sum-of-disjoint products (SDP) expression for Equation (5.1) which means that the results may also be used to identify local *disjoint subcubes*. Here, a disjoint subcube is defined as a subcube which consists of different nodes than those of any other subcube. The disjoint subcube is generated from the local node v and a disjoint term in the SDP expression by complementing the ith bit in the address of v when the disjoint term contains an x_i, and replacing some other bits in the address of v following the discussion in Section 5.3.2.1. However, one may use a simple transformation procedure given in [45] to interpret the SDP expression for sum-of-products expression. Note, in our description of the CMB operator, an independent (dependent) $\overline{FB_i}$ is represented as IG_i (DG_i).

The CMB+ operator is different from the CMB operator only on Step 2 used in **CASE 2** [CMB for dependent group] (refer to Section 3.3.2). Let STG_i denote an element of a stack STG. Initially, the stack STG contains the two elements created in Step 1 of the CMB operator (refer to Section 3.3.2). An STG_i contains a TG that

has not been CoMBined with the DG_k's, for $i \leq k \leq \rho$, where ρ is the total number of non-redundant DG's. Notice that the DG's are numbered from 1 through ρ. Step 2 for CMB+ is shown as follows.

CASE 2 [CMB for dependent group] /* Step 2 for CMB+ */
while ($STG \neq null$) **begin**
  pop (STG_i);
  DG (TG_i, DG_i, STG'_{i+1});
  /* STG'_{i+1} may have more than one terms */
  if ($i < \rho$)
    push (STG'_{i+1});
  else begin /*we have obtained some local OS's. */
    Obtain local OS's; /* include result from *independent* group */
    if (the subcube size is sufficient)
      return;
  end;
end;

The function **DG** is exactly the same as the one given in Section 3.3.2 except for the output STG'_{i+1} replacing TG'. When CMB+ is used exhaustively (until STG is empty), it gives exactly the same results as produced by the original CMB operator.

The proposed CMB+ provides two options for LOS identification. In the first option, the algorithm uses the operator exhaustively to generate local LOS(s). In the second option, the method utilizes the operator to identify some local OS(s) at each step. Here, the algorithm may stop searching for more local OS's when the identified local OS is sufficient for system reconfiguration (even though the OS is not a LOS). Note, the CMB operator always provides exhaustive results and, hence,

is less general than CMB+. In what follows, we present the proposed distributed LOS identification technique called LOS1. Links are assumed to be reliable; however, the algorithm may also be extended for unreliable links. The dependent elements of FB_i's may be ordered by their increasing cardinality so that the operator can complete this step faster. Refer to reference [75] for a discussion on the advantage of using this ordering.

```
Algorithm LOS1 /*Input F_i's and local node v */
begin
  /* This step generates non-redundant FD_i's */
  for (all F_i's) begin
    Generate FD_i from v and F_i;
    /* for all independent and dependent FD_j's */
    if (FD_i ⊆ FD_j)
      Remove FD_j;      /* FD_j is a redundant term */
    else if (FD_j ⊆ FD_i)
      Remove FD_i;      /* FD_i is a redundant term */
    else
      Put FD_i in a independent or dependent FD list;
  end;
  Order the dependent group FD_i's; /* and hence, the FB_i's */
  Use CMB+ operator on the FB_i's; /* independent and dependent */
end.
```

5.3.2.3 Illustrating Examples

Example 5.3. Consider a C_4, v = (0011), and a faulty node F_0. The LOS1 computes FD_0 = {0,1}, and hence, FB_0 is (x_1x_0). Since there is only one FB_i, the CMB operator is not needed for Equation (5.1). The node v and the $\overline{FB_0}$ give two local LOS's as (xxx1) and (xx1x).

Example 5.4. Figure 5.6 shows a C_4 with its local node v = (0000) and faulty nodes F_3 and F_9. Using the information, we generate dependent $\overline{FB_3} = \overline{x_1x_0}$ and $\overline{FB_9} = \overline{x_3x_0}$. The CMB+ operator generates $(\overline{x_1x_0})\ (\overline{x_3x_0}) = \overline{x_0} + \overline{x_3}\ \overline{x_1}x_0$. The SDP expression can be interpreted as such, to represent two disjoint subcubes (xxx0) and (0x01) (denoted by a "1" and a "3" in the figure). The expression, however, is transformed into the sum-of-products expression by deleting any non-inverted Boolean variables. For the example, two subcubes (xxx0) and (0x0x) (denoted by a "1" and a "2" in the figure) are generated. Note, the 3-subcube (xxx0) is a local LOS.

Example 5.5. Consider a C_6 with its local node v = (110011) and its four faulty nodes (010011), (110110), (101111), and (100101). An independent $\overline{FB_{19}} = \overline{x_5}$, and dependent FB's as: $\overline{x_2x_0}$, $\overline{x_4x_3x_2}$, $\overline{x_4x_2x_1}$ are obtained. Using the CMB+ operator, we solve the dependent FB's for (5.1) as

$(\overline{x_2x_0})(\overline{x_4x_3x_2})\ (\overline{x_4x_2x_1}) = (\overline{x_2} + \overline{x_4x_3x_2x_0})\ (\overline{x_4x_2x_1}) = (\overline{x_2}) + (\overline{x_4x_3x_2x_0})\ (\overline{x_4x_2x_1})$.

The first result, $\overline{x_2}$, and independent $FB_{19} = \overline{x_5}$ generate a 4-cube $\overline{x_5}\ \overline{x_2}$ = (1xx0xx). If the 4-cube is sufficient for system reconfiguration, we stop the algorithm. Otherwise, we let the method generate more OS's as: $(\overline{x_4x_3x_2x_0})(\overline{x_4x_2x_1}) = \overline{x_4}\ x_2\ \overline{x_0} + x_4\ \overline{x_3}\ x_2\ \overline{x_1}\ \overline{x_0}$. Thus, the method also generates a 3-cube $\overline{x_5}\ \overline{x_4}\ \overline{x_0}$ = (11xxx1) and a 2-cube $\overline{x_5}\ \overline{x_3}\ \overline{x_1}\ \overline{x_0}$ = (1x0x11). The first generated OS (1xx0xx) is a local LOS.

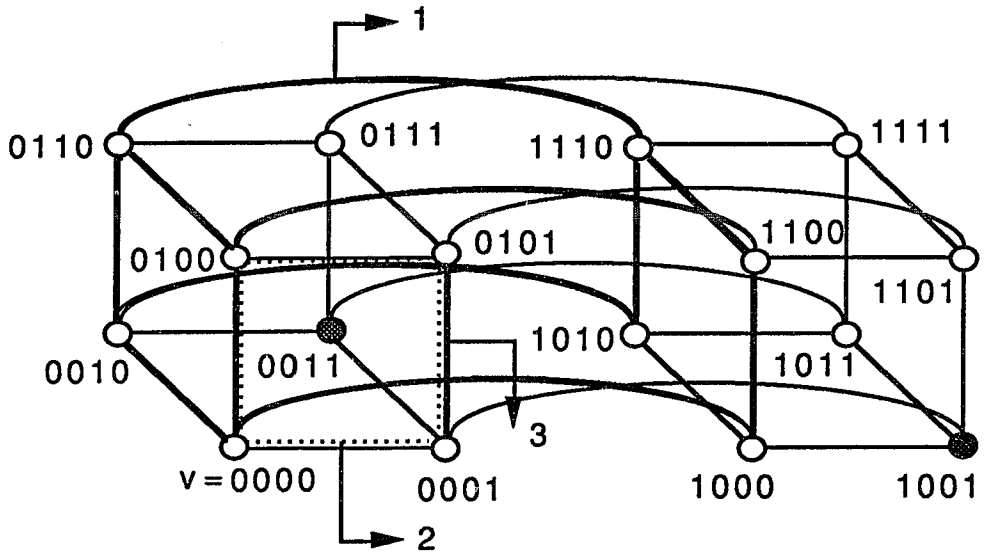


Figure 5.6. An Illustrating Example.

In the following example, we show that algorithm LOS1 performs better than the Greedy approach given in reference [48]. For this example, the Greedy method does not generate a local LOS (its largest OS is a 3-cube). Our approach, on the other hand, generates more than one possible subcubes, including a LOS.

Example 5.6. Consider a 7-cube, v = 0000000, and faulty nodes, F_3, F_5, F_9, F_{18}, F_{36}, and F_{72}. The LOS1 generates dependent FB_i's as: x_1x_0, x_2x_0, x_3x_0, x_4x_1, x_5x_2, and x_6x_3. The CMB+ operator computes

$(\overline{x_1x_0})\ (\overline{x_2x_0})\ (\overline{x_3x_0})\ (\overline{x_4x_1})\ (\overline{x_5x_2})\ (\overline{x_6x_3})$

$= (\overline{x_0} + \overline{x_2}\ \overline{x_1}x_0)\ (\overline{x_3x_0})\ (\overline{x_4x_1})\ (\overline{x_5x_2})\ (\overline{x_6x_3})$.

Combining the term $\overline{x_0}$ with the remaining four FB_i's, we obtain $\overline{x_6x_3}\ \overline{x_5x_2}\ \overline{x_4x_1}\ \overline{x_0}$, which represents eight 3-cubes (0xxx000), (x0x0x00), (00xxx00), (xx000x0), (0x0x0x0), (x000xx0), (000xxx0), and (xxx0000). The results are not optimal. Nonetheless, if the 3-cubes have met the request for reconfiguring the hypercube, we stop here. Otherwise, the algorithm will search for the other possible subcubes. For this example, the algorithm generates a local LOS $\overline{x_3}\ \overline{x_2}\ \overline{x_1} \equiv$ (xxx000x).

5.3.2.4 Comparison with the Existing Techniques.

Reference [48] presents two heuristic distributed algorithms for LOS identification. However, the methods do not always obtain a local LOS. Furthermore, both techniques generate only one local LOS. Reference [18] provides two algorithms, **A** and **B**, which always obtain at least one local LOS. Identifying more than one LOS is preferable since the supervisory node may have more flexibility in reconfiguring the faulty system. Thus, the algorithms in [18] are better and more general than those given in [48]. Algorithm **B** is faster than algorithm **A** [18] and, thus, the performance of our algorithm LOS1 is compared with that of algorithm **B**.

Both algorithms LOS1 and **B** generate at least one local LOS. From a local node and faulty nodes information, algorithms LOS1 and **B** obtain a set of non-redundant FD_i's and a set of non-redundant reject regions R, respectively. Generating the set FD is bit implementable since an FD_i is represented by symbols {0, 1}. On the other hand, a reject region is denoted by symbols {0, 1, x} and is more complex and slower to compute. Furthermore, redundancy checking for each FD_i (discussed in Section 5.3.2.7) is faster than that for the reject region. Obtaining the sum-of-products equivalents for $\overline{FB}$ and $\bar{R}$ is the most time consuming part of the algorithms LOS1 and **B**, respectively. Unfortunately, the methods used in both algorithms are heavily data dependent which makes it impossible to compare them using formal time complexity analysis. Furthermore, reference [18] does not provide any benchmark faulty node sets for its results, which otherwise may be used for comparing the speed of the two algorithms. The computational performance of algorithms LOS1 and **B**, however, depends on the number of Boolean product terms generated, so, we use the total number of the product terms in the two approaches as the parameter to compare their speeds. Note, this parameter is used for comparing the performance of Boolean techniques for the network reliability problem [33,74,85,86], which is similar to generating the sum-of-products here. Refer to Section 3.2. In what follows, we first show that LOS1 is expected to produce fewer Boolean product terms than algorithm **B**. Then, we use some illustrating examples to support the result.

In Section 3.2, we have discussed the concepts of single-variable inversion (SVI) and multiple-variable inversion (MVI) which are extensively used in Boolean techniques for the network reliability problem [86]. References [74,85,86] have shown that Boolean techniques based on the MVI concept are faster and generate fewer SDP terms than SVI-based techniques. Generating the sum-of-products

equivalent for $\overline{FB}$ in LOS1 and $\overline{R}$ in algorithm **B**, conceptually, is similar to the Boolean technique approach for the network reliability problem. The algorithm **B** utilizes the distributive law on Boolean expressions using the SVI notation. On the other hand, the CMB+ operator in LOS1 uses the new subcube notation to implement the MVI concept. Thus, LOS1 is expected to produce fewer terms and, hence, is faster than the algorithm **B**. Furthermore, when all FD_i's belong to the independent group, LOS1 solves the problem in polynomial time (in the number of faulty nodes). On the other hand, algorithm **B** does not solve this case as straightforwardly. To illustrate algorithms LOS1 and **B**, we provide the following examples.

Example 5.7. Consider a local node (1000), and the two faulty nodes (0001) and (0110). Algorithm **B** generates reject regions $R = \overline{x_3}x_0 + \overline{x_3}x_2x_1$. Thus, $P = \overline{R} = (x_3+\overline{x_0})\ (x_3+\overline{x_2}+\overline{x_1})$. Using distributive law and Boolean simplification, the algorithm obtains three terms as $x_3 + \overline{x_1}\ \overline{x_0} + \overline{x_2}\ \overline{x_0}$. On the other hand, LOS1 obtains $\overline{FB} = (\overline{x_3x_0})\ (\overline{x_3x_2x_1})$, and the CMB+ operator generates two terms $\overline{x_3} + x_3\overline{x_2x_1}\ \overline{x_0}$. Note, both algorithms identify exactly the same subcubes, but with different notations.

Example 5.8. Consider a C_7 with local node (0000000), and six faulty PE's (0000011), (0000101), (0001001), (0010010), (0100100), and (1001000). Algorithm **B** generates eight operational subcubes denoted in eight Boolean terms as $x_3x_2x_1 + x_6x_2x_1x_0 + x_5x_3x_1x_0 + x_6x_5x_1x_0 + x_4x_3x_2x_0 + x_6x_4x_2x_0 + x_5x_4x_3x_0 + x_6x_5x_4x_0$. On the other hand, LOS1 obtains the same subcubes, which are represented in more concise notations as $\overline{x_3}\ \overline{x_2}\ \overline{x_1x_0} + \overline{x_6x_3}\ \overline{x_5x_2}\ \overline{x_4x_1}\ \overline{x_0}$.

Example 5.9. In this example, the case for independent group is illustrated. Consider a C_8 with its local node v = (00000000) and four faulty nodes (00000011), (00001100), (00110000), and (11000000). Algorithm **B** obtains sixteen subcubes represented in sixteen Boolean terms as: $x_6x_4x_2x_0 + x_7x_4x_2x_0 + x_6x_5x_2x_0 + x_7x_5x_2x_0 + x_6x_4x_3x_0 + x_7x_4x_3x_0 + x_6x_5x_3x_0 + x_7x_5x_3x_0 + x_6x_4x_2x_1 + x_7x_4x_2x_1 + x_6x_5x_2x_1 + x_7x_5x_2x_1 + x_6x_4x_3x_1 + x_7x_4x_3x_1 + x_6x_5x_3x_1 + x_7x_5x_3x_1$. On the other hand, the LOS1 generates the same sixteen subcubes represented by only one Boolean term $\overline{x_7x_6}\ \overline{x_5x_4}\ \overline{x_3x_2}\ \overline{x_1x_0}$.

5.3.2.5 Efficient Centralized Approach

The algorithm LOS1 can also be used to identify the global LOS in an n-cube. To obtain the global LOS, one may run the algorithm LOS1 on every healthy node v, each of which generates local LOS(s). Then, the global LOS is obtained by selecting the largest local LOS. The approach, however, is not efficient since there are potentially $2^n - r$ healthy nodes in a C_n with r faulty nodes. To reduce the computational time complexity of the approach, we select a certain number of healthy nodes that are expected to give the global LOS. Such selected nodes are called *candidate* nodes. In this section, we consider two different approaches for generating the candidate nodes:

1) Use the healthy n neighbors of each faulty node as the candidate nodes [18]. This approach uses $O(nr)$ candidate nodes in C_n with r faulty nodes.
2) When there is a pair of antipodal faulty nodes, select the neighbors of either one of the two faulty nodes as the candidate nodes. However, if all antipodal nodes of the faulty nodes are healthy, use the r antipodal nodes as the candidate nodes. This approach uses $O(r)$ candidate nodes for a C_n with r faulty nodes.

Both methods 1) and 2) expect that the global LOS is generated from the candidate nodes. Approach 1) considers each candidate node as the *border* of the local LOS generated since if the cube includes the node beyond the border (a faulty node), the generated LOS is faulty. Approach 2), on the other hand, considers two different scenarios. The first scenario assumes that all the antipodals of the faulty nodes are healthy and selects them as the candidate nodes. Here, any of the healthy antipodal nodes are expected to obtain the global LOS since the faulty nodes are at the farthest distances from the candidate nodes. In the second scenario, the faulty nodes do not have all of their antipodal nodes healthy, i.e., there is at least one pair of antipodal faulty nodes. Note, two antipodal nodes are at the farthest distance from each other, and when the two nodes are faulty, they destroy all healthy $(n-1)$-cubes in an n-cube. Although approaches 1) and 2) do not always obtain global LOS, we use empirical results to show that the approaches help generate global LOS with high probability for C_n with n faulty nodes. In what follows, the performances of the two different sets of candidate nodes are compared using algorithm LOS1.

5.3.2.6 Experimental Results

For every randomly generated set of n faulty nodes in an n-cube, we obtain two sets of candidate nodes using approaches 1) and 2). Algorithm LOS1 is used on both sets to generate the local LOS from each candidate node. Then, the maximum sized local LOS is selected as a global LOS. For $n = 4, 5, \cdots, 16$, 3,000 sets of n faulty nodes have been randomly generated, where each 1,000 sets have one of the following properties:

(1) Each set has no pair of antipodal faulty nodes.

(2) Each set has only one pair of antipodal faulty nodes.

(3) No restrictions.

An exhaustive method was used to check the optimality and correctness of every local operational subcube (OS) generated. From about the 400,000 results, the obtained OS's were found to be correct and optimal. Furthermore, the sizes of the global LOS's generated using the two different sets of candidate nodes were found to be equal. Thus, approach 2) is better than approach 1) since the former produces fewer candidate nodes than the latter. A method given in [56] was used to verify the correctness of the global LOS's generated, and two non global LOS's were found out of 39,000 results. Although the proposed method does not always obtain global LOS's, the technique is preferable (than the previous centralized techniques) since it is faster and generates the optimal results with very high probability.

5.3.2.7 Computational Complexity of LOS1

Consider a C_n with r faulty nodes, and let v (F_i) be a local (faulty) node. We use bitwise *xor* on v and F_i to obtain FD_i, and hence FB_i. Bitwise *xor* is done in constant time, therefore, the FD_i's are generated in $O(r)$ steps. A redundant FD_j with respect to FD_i is detected using a bitwise *xor* and a bitwise *or* operation (Refer to a method discussed in Section 3.2). Considering r FD_i's, redundancy checking is done on $1 + 2 + 3 + \cdots + r-1 = \frac{r\ (r-1)}{2}$ pairs of FD_i and FD_j, and is completed in $O(r^2)$ steps. Note, the cardinality ordering on the non-redundant FD_i's can be overlapped with the redundancy checking steps so that no extra cost is added on the time complexity.

The computational complexity of the CMB+ operator is heavily data dependent, and, hence, we consider the best and the worst cases. The best case for the

operator is when $FD_i \cap FD_j = \emptyset$, for all $i \neq j$. Refering to Section 3.3.2, the FD_i's can be viewed as forming an independent group (IG). The CMB+ operator for the IG case is completed in $O(r)$ time. Therefore, LOS1 has $O(r^2)$ steps for the best case. For the worst case scenario, we use empirical results to show the largest number of times the CMB+ operator calls the **DG** function, and the largest number of times the **while loop** inside the function is executed. Note, the code in the **while loop** of the **DG** function is run in $O(n)$ steps. Table 5.4 shows the maximum number of steps involved in the CMB+ operator for C_n with n faulty nodes, for $n = 4, 5, \cdots, 16$. The table shows that for the worst case, the CMB+ operator uses the **DG** function $O(n^2)$ times, and the function executes the **while loop** $O(n)$ times. Thus, for C_n with r faulty nodes, we conjecture that the expected time complexity of algorithm LOS1 is no worse than $O(n^2r^2)$.

5.3.2.8 Polynomial Time Approach

The proposed algorithm LOS1 may also be used to generate local LOS(s) more efficiently by incorporating a greedy approach. This heuristic approach, however, requires the FB_i's (the input) to be ordered to help the CMB+ operator produce LOS(s) with high probability. For this greedy technique, the CMB+ operator stops generating the remaining results once it has obtained the first local OS(s), and the local OS(s) is considered as the local LOS(s). Here, CMB+ calls function **DG** only $\rho < n$ times, where ρ denotes the number of dependent FB_i's. Thus, the heuristic approach is completed in $O(r^2)$ steps for C_n with r faulty nodes. In what follows, we describe how to order the FB_i's. Note, this ordering is only performed once and is completed in polynomial time (in the order of the number of FB_i's).

A Boolean variable x_j is defined to have a *weight* α if there are α dependent group FB_i's that contain x_j. An FB_i is said to have a degree β if the maximum

Table 5.4.
The Experimental Results for Time Complexity of CMB+ Operator

C_n	#DG calls	#while loop used	Total #steps
4	3	2	6
5	6	3	18
6	12	4	24
7	23	4	92
8	38	5	190
9	61	5	305
10	94	5	490
11	175	5	875
12	250	5	1250
13	371	6	2226
14	645	6	3870
15	930	6	5580
16	1558	6	9348

weight of its variables is β. The non-redundant dependent FB_i's are sorted based on their decreasing degrees. If two or more FB_i's have the same degree, they are ordered following their increasing cardinality. However, if the same degree FB_i's are of the same cardinality, they may be ordered randomly. This ordering makes the CMB+ generate local OS's containing variables with larger weights.

Example 5.10. Consider $FB_1 = x_2x_1x_0$, $FB_2 = x_3x_2x_0$, $FB_3 = x_4x_0$, and $FB_4 = x_5x_4$. The variables x_0, x_1, x_2, x_3, x_4, and x_5 have weights 3, 1, 2, 1, 2, and 1, respectively, and the FB_1, FB_2, FB_3, and FB_4 have degrees 3, 3, 3, and 2, respectively. Thus, the FB_i's are ordered as FB_3, FB_1, FB_2, FB_4.

We have used the method to obtain a global LOS of a C_n using n candidate nodes (refer to Section 5.3.2.5.). First, 3,000 sets of n-faulty nodes for $n = 4, 5, \cdots, 16$ are randomly generated. Then, the heuristic algorithm is used to generate the local LOS's from the candidate nodes, and the largest subcube is selected as the global LOS. We found that the heuristic method identifies global LOS's more than 99% of the time. Furthermore, the dimensions of generated OS's (which are not LOS) are only one less than those of the optimal results. Since the method uses $O(n)$ local nodes, it is obvious that the algorithm takes $O(n^3)$ steps for generating global LOS (s) in a C_n with n faulty nodes.

5.3.3 Distributed Subcube Identification with Residual Nodes

In this section, we propose an algorithm called LOS2 which identifies $SC_\kappa(v)$ from a node v and a set of residual nodes W. For each node i, $w_i \in W$, the algorithm generates an SD_i. The SD_i contains a set of dimensions that provide a path from nodes v to w_i. An $SC_k(v)$ which includes node w_i is generated when node v is expanded towards all dimensional directions in SD_i. Note, a κ-cube of v has $\begin{bmatrix} \kappa \\ 1 \end{bmatrix}$

nodes which are 1-distant, $\binom{\kappa}{2}$ nodes which are 2-distant, $\cdots$, $\binom{\kappa}{\kappa-1}$ nodes which are κ–1-distant, $\binom{\kappa}{\kappa}$ node which is κ-distant from node v. This property is stated in the following lemma.

Lemma 5.1. For $i = 1, 2, \cdots, k$, an $SC_k(v)$ contains $\binom{k}{i}$ nodes, each of which is i Hamming distance from node v.

Our proposed algorithm uses Lemma 5.1 to identify a local LOS. First, the algorithm computes each SD_i by taking bitwise *xor* on the addresses of nodes v and w_i. Here, an SD_i is represented in bit notation and is interpreted by considering a "1" in a jth position as a dimensional direction j. Let $| SD_i |$ represent the number of 1's in SD_i, and let H_k be the set of SD_i's with k dimensions, i.e., $k = | SD_i |$. The method, then, groups the SD_i's into H_k's, for $k = 1, 2, \cdots, \eta$ and $\eta \leq n$. The algorithm searches for a local LOS using the SD_i's, starting from those in H_η, then those in $H_{\eta-1}$, $\cdots$, until a subcube is found. For each SD_i in H_k, the method checks if it has $\binom{k}{l}$ SD_j's in H_l as its subsets, for $l = k-1, \cdots, 2, 1$. If all such subsets exist, the SD_i is used to identify an $SC_k(v)$. Note, an SD_i of set H_k represents a node w_i that is k distant from v, and hence if the SD_i has all the required subsets, it also denotes a k-subcube. The k-subcube is obtained from the SD_i by replacing a "1" in SD_i with an "x". To reduce the computational complexity of the algorithm, the size, κ, of the subcube is computed so that the algorithm starts searching for the subcube from SD_i's in H_κ. However, we conjecture that computing exact κ is difficult, therefore a polynomial time method (in the number of working nodes) is used to obtain its upper bound, $d_{\max}$. The $d_{\max}$, in turn, is used to a

priorily remove nodes that are not in the subcube. In what follows, we provide lemmas which are used in our method. Lemmas 5.2 and 5.3 compute the $d_{\max}$, while Lemma 5.4 further removes the *unused* nodes.

Lemma 5.2. For $k = 1, 2, \cdots, n$, and where ω represents the first empty H_k,

$$d_{\max} \leq \min \{ \lfloor \log_2 |W| \rfloor , \omega - 1 \}.$$

Lemma 5.3. For $j = 1, 2, \cdots, k-1$, if $|H_j| < \begin{bmatrix} k \\ j \end{bmatrix}$ then $d_{\max} < k$.

Proof: When $|H_j| < \begin{bmatrix} k \\ j \end{bmatrix}$, k-subcube does not exist. □

Lemma 5.4. Node w_i is not in an $SC_\kappa(v)$ if $SD_i \not\subseteq \bigcup_{SD_j \in H_1} SD_j$.

Proof: Each node in an $SC_\kappa(v)$ is reachable from node v through the dimensional directions in H_1. □

Using Lemmas 5.1 through 5.4, we present the proposed algorithm LOS2.

```
Algorithm LOS2 /* Input: Set W and node v, Output: SC_κ(v) */
begin
  d_max = generate_Hk (v, W);
  for (k = d_max down to 1) begin
    for (SD_i ∈ H_k) begin
      if (subcube (SD_i, k))
        Print the local LOS;  /* We may stop the algorithm */
    end;
  end;
end.
```

Function **generate_Hk** computes SD_i's, groups them into H_k's, for $k = 1, 2, \cdots, d_{max}$, and returns the d_{max}. The subroutine utilizes Lemmas 5.2 through 5.4 and is implemented as follows.

```
generate_Hk (v, W)
begin
/* Implementing Lemma 5.2, and generating H_j's */
  d'_max = ⌊log2|W|⌋;
  for (w_i ∈ W) begin
    SD_i = v ⊕ w_i;
    if (k = |SD_i| ≤ d'_max)
      Put SD_i in set H_k;
  end;
```

for(k = 1 to d'_{max}) **begin**

if ($|H_k| \equiv 0$)

$d'_{max} = k-1$;

end;

/* Lemma 5.3 implementation */

for ($k = d'_{max}$ **down to** 2) **begin**

for ($j = k-1$ **down to** 1) **begin**

if ($|H_j| < \begin{bmatrix} k \\ j \end{bmatrix}$)

$d'_{max} = k-1$;

end;

end;

/* Lemma 5.4 implementation */

$SH = \bigcup_{SD_j \in H_1} SD_j$;

for(all SD_i's) **begin**

if ($SD_i \not\subseteq SH$)

Delete SD_i;

end;

return (d'_{max});

end;

Function **subcube** is based on Lemma 5.1, and returns *true* if the SD_i gives a subcube of size k. The function is implemented as follows.

subcube (SD_i, k)
begin
  for ($j = k-1$ **down to** 1) **begin**
    if (there are less than $\begin{bmatrix} k \\ j \end{bmatrix}$ subsets of SD_i in H_j)
      return (*false*);
  end;
  return (*true*);
end;

In what follows, we provide some illustrating examples to show the salient features of the proposed algorithm LOS2.

Example 5.11. Consider a C_4 with W = { (0000), (0001), (0101), (0110), (0111), (1000), (1001) } and local node v = (0001). Since $\lfloor \log_2 |W| \rfloor \equiv 2$, H_3 is not generated, and function **generate_Hk** obtains

H_1 = { (0001), (0100), (1000) },

H_2 = { (0110), (1001) }.

By Lemma 5.4, (0110) is deleted. Since (1001) has two subsets in H_1, (0001) and (1000), the algorithm identifies a 2-subcube (x00x).

Example 5.12. For a C_5, consider its eleven residual nodes: (00001), (01000), (01001), (10000), (10001), (10100), (10101), (11000), (11001), (11100), and (11101). Also consider that a local node v = (10001). Function **generate_Hk** creates

H_1 = { (00001), (00100), (01000), (10000) },

H_2 = { (00101), (01001), (01100), (11000) },

H_3 = { (01101), (11001) },

and $d_{\max}$ = 3. Function **subcube** ((11001), 3) returns *fail* since there are only 2 (less than $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$) subsets of (11001) in H_2. Thus, the method continues searching for the LOS by executing function **subcube** ((01101), 3). The routine returns *true* since there are $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and $\begin{bmatrix} 3 \\ 1 \end{bmatrix}$ subsets of (01101) in H_2 and H_1, respectively. Thus, the algorithm identifies a 3-cube (1xx0x).

5.3.3.1 Computational Complexity of LOS2

Let $m \leq 2^n$ denote the total number of residual nodes in a hypercube C_n, and $m_i = |H_i|$. The function **generate_Hk** obtains SD_j's, groups them into H_i's, and computes $d_{\max}$. Note, the LOS2 utilizes a bit vector representation [4] for local node v and for the residual nodes w_i's, and, hence, each SD_j is computed in one *xor* instruction. Since $|SD_j|$ is computed in n steps, this phase needs $O(mn)$ steps. The function implements Lemmas 5.3 and 5.4 in $O(n^2)$ and $O(m)$ steps, respectively. Thus, **generate_Hk** requires $O(mn)$ steps. On the other hand, function **subcube** takes $O(M_{1,d_{\max}-1})$ steps, for $M_{x,y} \equiv \sum_{i=x}^{y} m_i$. However, the total number of function calls is data dependent. For the best case, we consider $d_{\max} = \kappa$, and, therefore, the function is called at most m_κ times. When $d_{\max}$ is not tight, the total

number of function **subcube** calls is $O(M_{\kappa,d_{max}})$ times. Thus, the time complexity of the LOS2 is $O(\eta)$, for $\eta = \max\ (mn, M_{1,d_{max}-1} \cdot M_{\kappa,d_{max}})$.

5.3.3.2 Using LOS2 for a Centralized Approach

The algorithm LOS2 can also be used to generate the global LOS. The supervisory node executes the algorithm and considers each residual node as a local node. This approach obtains a global LOS in $O(\eta m)$, which is comparable to $O(m^2 n)$ of the method presented in [79]. However, our method is preferable since the supervisory node can stop the algorithm once it produces the required sized subcube. Furthermore, global LOS may be generated before the algorithm is executed for all possible local nodes, so, consequently, our proposed method is expected to obtain global LOS faster than the method in [79].

5.3.4 Conclusions

We have discussed methods for generating the largest operational subcubes for hypercubes with faulty components. The techniques can be categorized into centralized and distributed approaches. The distributed techniques for LOS identification are preferred over the centralized approaches since the former are more general and more efficient than the latter. Also, the distributed approach can be used to obtain the global LOS. Any of the methods need either faulty or residual component information. We have presented a distributed technique LOS1 that requires faulty component information, and a distributed algorithm LOS2 which needs residual node information. Both techniques always generate optimal results. For C_n with n faulty nodes, LOS1 is empirically shown to run in polynomial time in the order of cube dimension. On the other hand, LOS2 obtains both local and global LOS's in polynomial time in the order of the number of the residual nodes and the cube

dimension. Thus, LOS1 (LOS2) is used when the hypercube has less (more) faulty nodes than residual nodes.

Chapter 6

Conclusions and Future Directions

This research has been carried out to obtain improved methods for the probabilistic and deterministic reliability evaluation of the hypercube architecture. Various reliability measures, namely, network-connected (NCF), 2-connected (2CF), task-based (TBF) and subcube (SF) functionalities are considered for both models. We have attempted to solve exact measures for the probabilistic model. An efficient algorithm, called CAREL, is presented and used to help compute various exact reliability measures. CAREL evaluates the probabilistic NCF measure for a hypercube in time polynomial in the number of spanning trees of the cube. We have used the shelling orderings property of the hypercube topology to obtain the result. However, the number of spanning trees in a hypercube increases exponentially in the order of the cube dimension. Even though CAREL has been succesfully utilized for analyzing various distributed network reliability parameters, it fails to effectively evaluate the probabilistic measures for large hypercubes. Thus, we are motivated to look for efficient methods for computing the lower and upper bounds on the reliability measures.

Since no known polynomial time algorithms exist for exact computation of probabilistic measures for the hypercube, this dissertation has presented algorithms to compute lower bounds under node and/or link failures. The resulting bounds can be computed at least as efficiently as bounds obtained by previous methods, and the

new bounds are tighter, hence closer to exact, than earlier bounds. We have presented polynomial-time algorithms to improve bounds on NCF, 2CF, and TBF reliability metrics. The dissertation has improved mainly the lower bounds on the probabilistic measures. The lower bound is appealing since it gives an indicator that the system is at least this reliable. Even though the upper bound measure is not as favorable as the lower bound, it provides a better insight on the system reliability compared to that of the approximation approach. An upper bound result gives an optimistic analysis on the system reliability and is a tool to check for the correctness of the approximation value. Furthermore, when the lower and upper bounds are tight, it is unnecessary to generate the exact results. Thus, better approaches to improve both lower and upper bounds on reliability measures are of significant value. In this dissertation, we have proposed efficient techniques to compute tighter lower bounds on NCF and 2CF measures for the hypercube than those obtained by existing techniques. For the 2CF measure, our approach considers both node and link failures. Further work on computing tighter upper bounds on these reliability measures may improve our understanding on reliability aspects of the hypercube architecture. We have also proposed an efficient method to compute an upper bound on TBF measure which is better than existing approaches in terms of time complexity and the computed results. The existing methods generate only approximate results. A lower bound on this task-based measure, however, is left for future work. Another area in which work remains is to compute bounds on the probabilistic measures for the cube that account for dependence between component failures. A possible approach is to use the shock model, as was done by Trahan *et al.* [25] for multistage interconnection networks.

In this dissertation, we have discussed the deterministic model for reliability measures in hypercube. Various measures for this model are described. One

measure, reliable broadcasting, is a deterministic model with the NCF criterion. For a C_n, the existing reliable broadcasting techniques tolerate only up to $n-1$ faulty components. This limitation is due to the topological property of the hypercube. (The node connectivity of an n-cube is n.) However, in case the faulty components do not isolate any healthy node, broadcasting in the faulty hypercube is feasible even when there are more than $n-1$ faulty components. When the component failures are statistically independent, it is less likely that $n-1$ faulty components disconnect a node. Thus, implementing the forbidden fault set concept (discussed in Section 5.2) on reliable broadcasting will give a better practical result.

We have proposed two efficient distributed techniques to identify the largest operational subcubes in a faulty hypercube. The proposed methods are shown to be more efficient than existing methods. One of the proposed methods, LOS1, uses faulty node information for its input; it is also shown empirically to run in time polynomial in the order of the number of faulty nodes when there are up to n failed nodes. Furthermore, LOS1 produces more than one possible result, which gives the supervisory processor more flexibility in reconfiguring the system. The method can be used either exhaustively or heuristically. When LOS1 is used in the former case, it always generates local LOS(s). Heuristic LOS1, on the other hand, is shown experimentally to generate optimal results 99% of the time. Note that the heuristic LOS1 is a polynomial-time algorithm (in the order of the number of faulty nodes). The second proposed method, LOS2, generates a local LOS from the residual nodes. This method has a time complexity of $O(mn)$, where m is the number of available nodes. Both LOS1 and LOS2 can also be used as centralized approaches to generate the global LOS's. The distributed techniques for LOS identification are more efficient and more general than the centralized approaches.

The performability issue of the hypercube system is an equally important topic and presents a unified approach for the performance and reliability measures of the system. For a hypercube working in degraded mode, performability provides a deeper insight. This issue for the hypercube architecture has not received much attention. So, research should be done on the performability of a hypercube with the NCF, 2CF, TBF, and SF measures.

References

[1] J.A. Abraham, "An improved algorithm for network reliability," *IEEE Trans. Reliability, vol. R-28, pp. 58-61, Apr. 1979.*

[2] S. Abraham and K. Padmanabhan, "Reliability of the hypercube," *1988 International Conference on Parallel Processing*, pp. 90-94.

[3] R.M. Ahmed and J.L. Trahan, "Two-terminal reliability of hypercubes," *Proc. Southeastcon'91*, pp. 427-431, 1991.

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley, 1974.

[5] A. Al-Dhelaan and B. Bose, "Efficient fault tolerant broadcasting algorithm for the hypercube," *Proc. 4th Conf. on Hypercube Concurrent Computers and Applications*, pp. 123-128, 1989.

[6] A. Al-Dhelaan and B. Bose, "A new strategy for processor allocation in n-cube multiprocessor," *Proc. Int. Phoenix Conf. Comput. Commun.*, pp. 114-118, March 1989.

[7] J.R. Armstrong and F.G Gray, "Fault diagnosis in a Boolean n-cube array of multiprocessors," *IEEE Trans. Computers*, vol. 30, pp. 587-590, Aug. 1981.

[8] M.O. Ball and J.S. Provan, "Disjoint products and efficient computation of reliability," *Operations Research*, vol. 36, no. 5, pp. 703-715, 1988.

[9] B. Becker and H.U. Simon, "How robust is the n-cube ?," *Proc. 27th IEEE Symposium on Foundations of Computer Science*, pp. 283-291, Oct. 1987.

[10] Bell Communications Research, *Reliability Prediction Procedure for Electronic Equipment*, Sept. 1985.

[11] R.G. Bennetts, "Analysis of reliability block diagrams by Boolean techniques," *IEEE Trans. Reliability*, vol. R-31, pp. 159-166, June 1982.

[12] K.V. Bhatt, "An efficient approach for fault diagnosis in a Boolean n-cube array of microprocessors," *IEEE Trans. Computers*, vol. C-32, pp. 1070-1071, 1983.

[13] L.N. Bhuyan and D.P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Computers*, pp. 323-333, Apr. 1984.

[14] Z.W. Birnbaum, J.D. Esary, S.C. Saunders, "Multicomponent structures and their reliability," *Technometrics*, vol. 3, pp. 55-77, 1961.

[15] D. Bulka and J.B. Dugan, "A lower bound on the reliability of an n-dimensional hypercube," *Proc. 9th Symposium on Reliable Distributed Systems*, pp. 44-53, 1990.

[16] M.S. Chen and K.G. Shin, "Processor allocation in an n-cube multiprocessor using Gray codes," *IEEE Trans. Computers*, vol. C-36, pp. 1396-1407, Dec. 1987.

[17] M-S. Chen and K.G. Shin, "Depth first search approach for fault tolerant routing in hypercube multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 152-159, Apr. 1990.

[18] H. Chen and N. Tzeng, "Quick determination of subcubes in a faulty hypercube," Proc. International Conf. Parallel Processing, 1992.

[19] V.K.R. Chivuluri and I. Koren, "Reliability analysis of a highly-integrated multiprocessor system," *1992 IEEE Workshop on Fault Tolerant Parallel & Distributed Systems*, Amherst, MA, pp. 54-61, July 6-7, 1992.

[20] C.J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, New York, 1987.

[21] C.R. Das and J. Kim, "A unified task-based dependability model for hypercube computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 3, pp. 312-324, May 1992.

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

[23] E. Dilger and E. Amman, "System level self-diagnosis in n-cube connected multiprocessor," *Proc. IEEE 14th International Symp. on Fault Tolerant Computing*, pp. 184-189, 1984.

[24] S. Dutt and J.P. Hayes, "On allocating subcubes in hypercube multiprocessor," *Proc. 3rd Conf. Hypercube*, pp. 801-810, Jan. 1988.

[25] A. El-Amawy and R. Raja, "Split sequence generation algorithms for efficient identification of operational subcubes in faulty hypercubes," *Parallel Computing* (To appear).

[26] P. Erdos and J. Spencer, "Evolution of the n-cube," *Comp. & Maths. with Appls.*, vol. 5, pp. 33-39, 1979.

[27] A. Esfahanian, "Generalized measures of fault tolerance with application to n-cube networks," *IEEE Trans. Computers*, vol. 38, pp. 1586-1591, Nov. 1989.

[28] P. Fraigniaud and C. Peyrat, "Broadcasting in a hypercube when some calls fail," *Information Processing Letters*, no. 39, pp. 115-119, Aug. 1991.

[29] M.R. Garey and D.S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[30] J. Gray and D.P. Siewiorek, "High-availability computer systems," *IEEE Computer*, pp. 39-48, Sept. 1991.

[31] A. Grnarov, L. Kleinrock, and M. Gerla, "A new algorithm for network reliability computation," *Proc. Computer Networking Symp.*, pp. 17-20, Dec. 1979.

[32] F. Harary, J.P. Hayes, and H. Wu, "A survey of the theory of hypercube graphs," *Comput. Math. Applic.*, vol. 15, pp. 277-289, 1988.

[33] S. Hariri and C.S. Raghavendra, "SYREL: A symbolic reliability algorithm based on path cutset methods," *IEEE Trans. Computers*, C-36(10), pp. 1224-1232, Oct. 1987.

[34] J.P. Hayes, T. Mudge, Q.F. Stout, S. Colley, and J. Palmer, "A microprocessor-based hypercube supercomputer," *IEEE Micro*, pp. 6-17, Oct. 1986.

[35] J.P. Hayes and T. Mudge, "Hypercube supercomputers," *Proc. of the IEEE*, vol. 77, no. 12, pp. 1829-1841, Dec. 1989.

[36] K.D. Heidtmann, "Smaller sums of disjoint products by subproduct inversion," *IEEE Trans. Reliability*, vol. 38, pp. 305-311, August 1989.

[37] *Intel, Micropocessors, Volume I*, 1991

[38] *Intel, Memory Products*, 1992

[39] *Intel, Components Quality and Reliability*, 1992

[40] G.B. Jasmon and O.S. Kai, "A new technique in minimal path and cutset evaluation," *IEEE Trans. Reliability*, vol. 34, no. 2, pp. 136-143, Jun. 1985.

[41] S.L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Computers*, vol. 38, no. 9, pp. 1249-1268, Sept. 1989.

[42] J. Kim, C.R. Das, W. Lin, and T. Feng, "Reliability evaluation of hypercube multicomputers," *IEEE Trans. Reliability*, vol. 38, pp. 121-129, Apr. 1989.

[43] J. Kim, C.R. Das, and W. Lin, "A top-down processor allocation scheme for hypercube computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, no. 1, pp. 20-30, Jan. 1991.

[44] D.J. Klinger, Y. Nakada, and M.A. Menendez, *AT&T Reliability Manual*, Van Nostrand Reinhold, New York, 1990.

[45] D.Y. Koo, "D/Boolean application in reliability analysis," *1990 Proceedings Annual Reliability and Maintainability Symposium*, pp. 294-301.

[46] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Programming Languages Systems*, vol. 4, pp. 382-401, Jul. 1982.

[47] S. Latifi, "Hypercube-based topologies with incremental link redundancy," *Ph.D. Dissertation*, Louisiana State University, August 1989.

[48] S. Latifi, "Distributed subcube identification algorithms for reliable hypercubes," *Information Processing Letters*, vol. 38, pp. 315-321, 1991.

[49] S. Latifi, "Identification of operational subcubes in unreliable hypercubes," *IEE Proc-E*, vol. 139, no. 2, pp. 117-122, Mar. 1992.

[50] T.C. Lee and J.P. Hayes, "A fault-tolerant communication scheme for hypercube computers," *IEEE Trans. Computers*, vol. 41, no. 10, pp. 1242-1256, Oct. 1992.

[51] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufman Publishers, San Mateo, CA, 1992.

[52] *Military Handbook*, "Reliability prediction of electronic equipment," MIL-HDBK-217F, Dec. 1991.

[53] R. Miller, *Switching Theory, Vol. I: Combinational Circuits*, New York: Wiley, 1965.

[54] W. Najjar and J.L. Gaudiot, "Reliability and performance modeling of hypercube-based multiprocessors," *Computer Performance and Reliability* (edited by G. Iazeolla *et al.*), pp. 305-320, 1988.

[55] W. Najjar and J.L. Gaudiot, "Network resilience : A measure of network fault tolerance," *IEEE Trans. Computers*, vol. 39, no. 2, pp. 174-181, Feb. 1990.

[56] F. Ozguner and C. Aykanat, "A reconfiguration algorithm for fault tolerance in a hypercube multiprocessors," *Information Processing Letters*, vol. 29, pp. 247-254, 1988.

[57] F.R. Preparata and J. Vuillemin, "The cube connected cycles: A versatile network for parallel computation," *Communication ACM*, vol. 24, pp. 300-309, 1981.

[58] X. Qian, N. Shiratori, and S. Noguchi, "An algorithm for evaluating K-terminal reliability," *Private communication.*

[59] M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, New York: McGraw-Hill, 1987.

[60] S. Rai and K.K. Aggarwal, "An efficient method for reliability evaluation of a general network," *IEEE Trans. Reliability*, vol. R-27, pp. 206-211, Aug. 1978.

[61] S. Rai and K.K. Aggarwal, "On complementation of pathsets and cutsets," *IEEE Trans. Reliability*, vol. R-29, pp. 139-140, Jun. 1980.

[62] S. Rai and D.P. Agrawal, *Distributed Computing Network Reliability*, IEEE Computer Society Press, 1990. (Tutorial Text)

[63] S. Rai and S. Soh, "A computer approach for reliability evaluation of telecommunication networks with heterogeneous link-capacities," *IEEE Trans. Reliability*, pp. 441-451, Oct. 1991.

[64] S. Rai and J.L. Trahan, "Computing network reliability of redundant MINs," *Proc. 21st Southeastern Symp. Syst. Theory*, Tallahassee, FL, pp. 221-225, Mar. 1989.

[65] S. Rai and J.L. Trahan, "ATARIC: An algebraic technique to analyse reconfiguration for fault tolerance in a hypercube," *Proc. 1991 International Conference on Parallel Processing*, pp. 548-555, 1991.

[66] S. Rai and J.L. Trahan, "A reconfiguration technique for fault tolerance in a hypercube," *Parallel Processing Letters* (Accepted).

[67] P. Ramanathan and K.G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. Computers*, vol. 37, pp. 1654-1657, Dec. 1988.

[68] J. Rattner, "Concurrent processing: a new direction in scientific computing," *AFIPS Conference Proceedings*, vol. 54, pp. 157-166, 1985.

[69] D.A. Reed and R.M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, The MIT Press, Cambridge, MA, 1987.

[70] Y. Saad and M.H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Computers*, vol. 37, pp. 867-872, Jul. 1988.

[71] C.L. Seitz, "The cosmic cube," *Communications ACM*, vol. 28, pp. 22-33, Jan. 1985.

[72] D.R. Shier and D.E. Whited, "Algorithms for generating minimal cutsets by inversion," *IEEE Trans. Reliability*, vol. 34, no. 4, pp. 314-319, Oct. 1985.

[73] D.P. Siewiorek and R.W. Swarz, *Reliable Computer Systems: Design and Evaluation,*, Digital Press, Bedford, MA, 1992.

[74] S. Soh and S. Rai, "CAREL: Computer aided reliability evaluator for distributed computer systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 2 pp. 199-213, Apr. 1991.

[75] S. Soh and S. Rai, "Experimental results on preprocessing of path/cut terms in sum of disjoint products technique," *IEEE Trans. Reliability* (To appear in April 1993 issue).

[76] S. Soh and S. Rai, "A cutset approach to survivability evaluation of large telecommunication-networks with heterogeneous link-capacities," *Proc. International Symposium on System and Circuits*, June 1991.

[77] S. Soh, S. Rai, and J.L. Trahan, "Improved lower bounds on the reliability of hypercube architectures," *Fifth ISMM Int. Conf. on Parallel and Distributed Computing and Systems*, pp. 182-187, Oct. 1992.

[78] S. Soh, S. Rai, and J.L. Trahan, "Improved lower bounds on the reliability of hypercube architectures," *IEEE Trans. Parallel and Distributed Systems* (Under revision).

[79] M.A. Sridhar and C.S. Raghavendra, "On finding maximal subcubes in residual hypercubes," *Proc. 2nd IEEE Symposium on Parallel and Distributed Processing*, pp. 870-873, Dec. 1990.

[80] H. Sullivan, T. Bashkow, and D. Klappholz, "A large, homogeneous, fully distributed parallel machine," *Proc. 4th Annual Symposium on Computer Architecture*, pp. 105-124, March 1977.

[81] R.E. Tarjan, "A unified approach to path problems," *Journal ACM*, vol. 28, pp. 577-593, 1981.

[82] R.K. Tiwari and M. Verma, "An algebraic technique for reliability evaluation," *IEEE Trans. Reliability*, vol. R-29, pp. 311-313, Oct. 1980.

[83] J.L. Trahan, D.X. Wang, and S. Rai, "Incorporating dependent and multimode failures into reliability evaluation of extra stage shuffle-exchange MINs," *Proc. 30th Allerton Conf. Communication, Control, and Computing*, Univ. of Illinois, Monticello, IL, Oct. 1992.

[84] S. Tsukiyama, I. Shirakawa, and H. Ozaki, "An algorithm to enumerate all cutsets of a graph in linear time per cutset," *Journal ACM* , vol. 27, no. 4, pp. 619-632, Oct. 1980.

[85] M. Veeraraghavan and K. Trivedi, "An improved algorithm for symbolic reliability analysis," *IEEE Trans. Reliability*, vol. R-40, pp. 347-358, Aug. 1991.

[86] M. Veeraraghavan and K.S. Trivedi, "Techniques for two-terminal network reliability evaluation," *Private Communication.*

[87] J. Wu and E.B. Fernandez, "Reliable broadcasting in faulty hypercube computers," *Proc. 11th IEEE Symp. on Reliable Distributed Systems*, pp. 122-129, 1992.

[88] C.S. Yang, J.F. Wang, J.Y. Lee, and F.T. Boesch, "Graph theoretic reliability analysis for the Boolean n-cube networks," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1175-1179, Sept. 1988.

[89] S.Y. Yoon, O. Kang, H. Yoon, S.R. Maeng, and J.W. Cho, "A heuristic processor allocation strategy in hypercube systems," *Proc. Third IEEE Symposium on Parallel and Distributed Processing*, pp. 574-581, 1991.

Appendix A

Proving Correctness of CAREL

The CAREL uses four operators, namely COM, RED, CMB, and GEN to transform minimal path (or cut) identifiers into an equivalent exclusive and mutually disjoint (e.m.d.) expression. The COM operator defines conditional events E_j's for an F_i, while RED removes redundant E_j's to produce minimal conditional event (MCE). The operation CMB combines MCE to generate disjointing terms (DT's). The DT's are mutually disjoint. Moreover, the F_i and its DT's (refer to GEN operator) form expression which is disjoint with all other terms in Equation (3.1). In what follows, we show that the CAREL always generates e.m.d. terms (DT's) for an F_i.

Section 3.3.2 describes COM operator. A conditional event E_j considers 'α' elements of F_j which are not present in F_i. To represent E_j down (refer to Section 3.3.1), our notations use $-\alpha^I$ in the positions of the variable [here the index I is equal to the position of the first variable in E_j]. Considering CAREL (Section 3.4.1), it is obvious that COM obtains all possible E_j's for an F_i.

Lemma A.1. Assume two conditional events E_j and E_k of F_i. If $E_j \subseteq E_k$, then E_k is redundant. □

The RED operator implements Lemma A.1, and is performed for all (E_j, E_k) pairs. Note, the RED defined in Section 3.3.2 checks out $\bar{x}\ \bar{x} = \bar{x}$, and $\bar{x}\ (\bar{x}\ \bar{y}) = \bar{x}$ type of redundancies. The non-redundant E_j's form MCE. Besides removing

redundancies, the RED operator partitions the MCE into IG's and DG's (refer to Section 3.4.2). Thus, the RED speeds up the computation time of the CAREL. The CMB operator for independent group (IG) uses $\bar{x} * \bar{y} = \bar{x}\,\bar{y}$ iteratively. As is obvious from Theorem A.1 (discussed later), $\bar{x} * \bar{y}$ is a special case of $X_i^{c_i} * X_j^{c_j}$. Here, we assume that no common elements are present with x and y. [x and y are independent.] The CMB operator for dependent group (DG) utilizes lemmas and theorems mentioned below.

Definition. An X_i represents an term which could be an MCE E_i, or the one produced during CMB operation. For notational convenience, consider $X_i^c = X_i\ (\overline{X_i})$, when $c = 1\ (0)$.

Lemma A.2. Assume $T_1^i, T_2^i, \cdots, T_k^i$ represent k - partitions of X_i. Then,

$$X_i^c = \begin{cases} (T_1^i\ T_2^i\ \cdots\ T_k^i); & c=1 \\ (\overline{T_1^i}, T_1^i\ \overline{T_2^i}, \cdots, T_1^i T_2^i\ \cdots\ T_{k-1}^i \overline{T_k^i}); & c=0 \end{cases}$$

Proof: For $c = 1$, the result is obvious. With $c = 0$, De-Morgan's complementation law is rewritten so that the list of T_j^i's is collectively exhaustive. □

Theorem A.1. Consider $I, L\ (J, K)$ as 2-partitions of $X_i\ (X_j)$. We have :

$$X_i^{c_i} * X_j^{c_j} = (IL)^{c_i}\ (JK)^{c_j}\ .$$

The terms $(IL)^{c_i}\ (JK)^{c_j}$ is

a) $X_i^{c_i} X_j^{c_j}$; if X_i and X_j are independent.

b) A ; otherwise, i.e., $L = J$. Thus, $L(=J)$ represents a common term between X_i and X_j.

For various combinations of c_i and c_j, A is obtained as -

i) **Case 1.** $[c_i = c_j = 1]$. $A = I\ J\ K$.

ii) **Case 2.** $[c_i = 1(0), c_2 = 0(1)]$.

$$A = \begin{cases} \varnothing; & I, K = \varnothing \\ I^{c_i}\ J\ K^{c_j}; & I, K \neq \varnothing \end{cases}$$

iii) **Case 3.** $[c_i = c_j = 0]$. $A = (\bar{J}, I\ \bar{J}\ \bar{K})$

Proof: For $X_i \cap X_j = \varnothing$, $X_i^{c_i} * X_i^{c_j} = X_i^{c_i}\ X_i^{c_j}$ is straightforward. Use Lemma A.2 to show the results in Cases 1 through 3. With $c_i = c_j = 1$ and $L = J$, Case 1 is obvious. For $c_i = 1$, $c_j = 0$, $X_i^{c_i} * X_j^{c_j}$ is $(I\ J)\ (\overline{JK})$ or $(I\ J)\ (\bar{J}\ ,\ J\ \bar{K})$. A result $(\bar{J}, J\ \bar{I})\ (JK)$ follows similarly when $c_i = 0$, $c_j = 1$. Thus, Case 2 of the theorem is proved. For $c_i = c_j = 0$, $X_i^{c_i} * X_j^{c_j} = (\overline{IJ})\ (\overline{JK})$. Using Lemma A.2, we interpret this result as $(\bar{J}, J\ \bar{I})\ (\bar{J}, J\ \bar{K})$ which after applying a Boolean identity [20] produces $(\bar{J}, J\ \bar{I}\ \bar{K})$. □

Lemma A.3. Note, $X * \varnothing = \varnothing * X = \varnothing$, where $\varnothing$ is a null set, and X represents any term.

Proof: Using Theorem A.1, the proof is obvious. □

Theorem A.2. For X_1, X_2, X_3, the CMB operator produces -

$$X_1\ \bar{X}_2 * \bar{X}_3 = G\ F\ (\bar{H}, H\ \bar{I}\ \bar{J}),$$

where G,F (I,H) represent 2-partitions for X_1 (X_2), and F, H, J are 3-partitions for X_3.

Proof: Note, $X_1\ \bar{X}_2$ is a term obtained considering $X_i^{c_i} * X_j^{c_j}$ in Theorem A.1 and represent mutually independent terms i.e., X_1 and X_2 will have no term in common. Using Lemma A.1, $(X_1\ \overline{X_2} * \overline{X_3})$ is shown to be equal to $(G\ F\ \overline{IH}\ \overline{FHJ})$. Theorem A.2 is proved after applying Lemma A.2. □

From Theorems A.1 and A.2, it is clear that if we CMB k number of X_i's, we generate a term of the type $X_1^{c_1}\ X_2^{c_2}\ \cdots\ X_k^{c_k}$. An iterative application of these theorems solves -

$(X_1^{c_1}\ X_4^{c_4}\ X_5^{c_5}\ \cdots\ X_k^{c_k} * (X_2^{c_2}\ X_3^{c_3})$ as $(\ \cdots\ (X_1^{c_1} * X_2^{c_2}) * X_3^{c_3} * \cdots\ ) * X_k^{c_k}$

Note, a CMB obtains e.m.d. terms [we have called them as DT's].

Finally, GEN combines the F_i with the disjointing terms DT's. The operator utilizes five out of twelve possible combinations for ($-\beta^I$, 0, 1) alphabets and is demonstrated to be complete (refer to the text). Hence, the algorithm CAREL is proved to be correct.

Appendix B

Proofs of Lemmas 4.1 and 4.2

Since we consider only shortest paths from PE 0 to PE 2^n-1, each path is described by a list of n distinct integers from $\{0, 1, \cdots, n-1\}$. In the following, when we refer to paths, we are discussing only paths of this type. For a path A, let $A(i)$ denote the the ith element in the path description, for $0 \leq i \leq n-1$.

Definition. An *m-prefix* of a path is the ordered list of the first m positions in the path. (For a path A, this is positions $A(0), A(1), \cdots, A(m-1)$.) An *m-prefix set* of a path is the (unordered) set of elements in its *m-prefix*.

Observe that if, for some m, two paths A and B have the same *m-prefix* set, then the paths share a common PE at distance m. If, in addition, $A(m) = B(m)$, then the paths share a common CE. This observation leads to the following lemmas.

Lemma B.1. Two paths A and B share a PE if and only if there exists an m, where $0 \leq m \leq n-1$, such that the *m-prefix* set of A equals the *m-prefix* set of B.

Lemma B.2. Two paths A and B share a CE if and only if either (i) $A(0) = B(0)$ or (ii) there exists an m, where $1 \leq m \leq n-1$, such that the *m-prefix* set of A equals the *m-prefix* set of B and $A(m) = B(m)$.

From Algorithm PG, note that we can describe a path $P_{i,j}$ as shown below. This description will be useful in our proofs of the properties. Let $flip(x_1, x_2, \cdots, x_{k-1}, x_k) = (x_k, x_{k-1}, \cdots, x_2, x_1)$. Let $<x>_n$ denote x modulo n. For $i = 1$,

$$P_{1,j} = (j-1), <j>_n, \cdots, <j-2>_n, \tag{B.1}$$

and for $2 \leq i \leq \alpha$,

$$P_{i,j} = (j-1), (<i+j-1>_n, \cdots, <j-3>_n), flip(j, \cdots, <i+j-2>_n), <j-2>_n. \tag{B.2}$$

With such descriptions, the *m-prefix* set for $P_{i,j}$ where $1 \leq m \leq n-i$ is

$$\{j-1, <i+j-1>_n, \cdots, <i+j+m-3>_n\}, \tag{B.3}$$

and the *m-prefix* set where $n-i+1 \leq m \leq n-1$ is

$$\{j-1, <j-m-1>_n, \cdots, <j-3>_n\}. \tag{B.4}$$

In this appendix, we will first prove Lemma 4.2, then Lemma 4.1. We follow this order because the proofs for the properties for Lemma 4.1 (both PE's and CE's, smaller α) will follow closely after the proofs for the properties for Lemma 4.2 (CE's only, larger α).

Proof for Lemma 4.2 :

Property L1. Paths $P_{i,j}$ and $P_{i,l}$ are CE disjoint, for $j \neq l$.

Proof: *Property L1* is implied by *Property L3*, proved below, for $i = k$. □

Property L2. Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common CE's, for $i < k$.

Proof: Consider $i=1$ first. By Equations (B.1) and (B.2), $P_{1,j}(0) = P_{k,j}(0) = j-1$, so by Lemma B.1, the paths have at least this one CE in common.

We next establish that no m exists, where $2 \leq m \leq n-2$, such that the *m-prefix* sets of $P_{1,j}$ and $P_{k,j}$ are equal. For $2 \leq m \leq n-k$, the *m-prefix* set of $P_{1,j}$ is $\{j-1, <j>_n, \cdots, <j+m-2>_n\}$ and the *m-prefix* set of $P_{k,j}$ is $\{j-1, <k+j-1>_n, \cdots, <k+j+m-3>_n\}$. Since $k \neq 1$, these sets cannot be equal. For $n-k+1 \leq m \leq n-2$, the

m-prefix set of $P_{1,j}$ is $\{j-1, <j>_n, \cdots, <j+m-2>_n\}$ and the *m-prefix* set of $P_{k,j}$ is $\{j-1, <j-m-1>_n, \cdots, <j-3>_n\}$. Since $m < n-1$, these sets also cannot be equal.

The $(n-1)$-*prefix* sets of the two paths are equal, however, and $P_{1,j}(n-1) = P_{k,j}(n-1) = <j-2>_n$, so the paths share this CE, and $P_{1,j}$ and $P_{k,j}$ have exactly two common CE's.

Now consider $2 \le i \le n-2$. Clearly, $P_{i,j}(0) = P_{k,j}(0)$, so by Lemma B.2, the paths have at least this one CE in common.

We next establish that there is no m, where $2 \le m \le n-i-1$, such that the *m-prefix* sets of $P_{i,j}$ and $P_{k,j}$ are equal. Then we establish that the $(n-i)$-*prefix* sets of $P_{i,j}$ and $P_{k,j}$ are equal and that $P_{i,j}(g) = P_{k,j}(g)$ for $n-i \le g \le n-1$. Thus, $P_{i,j}$ and $P_{k,j}$ have exactly $(i+1)$ common CE's, for $i < k$.

For $2 \le m \le n-k$, the *m-prefix* set of $P_{i,j}$ is $\{j-1, <i+j-1>_n, \cdots, <i+j+m-3>_n\}$ and the *m-prefix* set of $P_{k,j}$ is $\{j-1, <k+j-1>_n, \cdots, <k+j+m-3>_n\}$. Since $i \neq k$, these sets cannot be equal. For $n-k+1 \le m \le n-i-1$, the *m-prefix* set of $P_{i,j}$ is $\{j-1, <i+j-1>_n, \cdots, <i+j+m-3>_n\}$ and the *m-prefix* set of $P_{k,j}$ is $\{j-1, <j-m-1>_n, \cdots, <j-3>_n\}$. Since $m < n-i$, these sets can not be equal.

The $(n-i)$-*prefix* set of $P_{i,j}$ is equal to the $(n-i)$-*prefix* set of $P_{k,j}$. (See Equation (B.4) and recall that $i < k$.) For $n-i \le g \le n-2$, $P_{i,j}(g) = P_{k,j}(g) = <j-g-2>_n$. For $g = n-1$, $P_{i,j}(n-1) = P_{k,j}(g) = <j-2>_n$. Therefore, the last i CE's in $P_{i,j}$ and $P_{k,j}$ are common, giving a total of $i+1$ common CE's. □

Property L3. Paths $P_{i,j}$ and $P_{k,l}$ are CE disjoint, for $j \neq l$.

Proof: We will establish for $j \neq l$ that $P_{i,j}(0) \neq P_{k,l}(0)$ and that no m exists such that $P_{i,j}$ and $P_{k,l}$ have equal *m-prefix* sets. Assume without loss of generality that $i \leq k$. Consequently, $1 \leq i \leq k \leq n-2$ and $1 \leq j, l \leq n$.

Our proof will be organized into five cases based on the value of m. The cases are as follows: $m = 1$, $m = 2$, $3 \leq m \leq n-k$, $n-k+1 \leq m \leq n-i$, and $n-i+1 \leq m \leq n-1$. The reason for the last three cases is that we characterize *m-prefix* sets for $P_{i,j}$ in one way for $1 \leq m \leq n-i$ and in another way for $n-i+1 \leq m \leq n-1$; similarly, we characterize *m-prefix* sets for $P_{k,l}$ in one way for $1 \leq m \leq n-k$ and in another way for $n-k+1 \leq m \leq n-1$. (See Equations (B.3) and (B.4).) For Cases 1, 3, 4, and 5, we prove that for no m in the given range can $P_{i,j}$ and $P_{k,l}$ have equal *m-prefix* sets. For Case 2, we show that, even though equal 2-prefix sets are possible, that the paths do not have the third CE in common.

Case 1: $m = 1$. Since $j \neq l$, $P_{i,j}(0) = j-1 \neq P_{k,l}(0) = l-1$.

Case 2: $m = 2$. For this case we will show that either the *2-prefix* sets of the paths are not equal or, if they are equal, then the paths do not share the following CE. The *2-prefix* sets are as follows: the *2-prefix* set for $P_{i,j}$ is $\{j-1, <i+j-1>_n\}$, and the *2-prefix* set for $P_{k,l}$ is $\{l-1, <k+l-1>_n\}$. Since $j \neq l$, the only way the two sets can be equal is if $j-1 = <k+l-1>_n$ and $l-1 = <i+j-1>_n$. This implies $<j>_n = <k+l>_n$ and $<l>_n = <i+j>_n$.

First, we assume that $j, k \neq n$. Because $j = <k+l>_n$, then either (i) $j = k+l$ or (ii) $j = k+l-n$, and because $l = <i+j>_n$, then either (iii) $l = i+j$ or (iv) $l = i+j-n$. For the *2-prefix* sets to be equal, one of (i) or (ii) and one of (iii) or (iv) must be true. Suppose (i) is true, that is $j = k+l$. Then (iii) cannot hold, as it implies $i+k = 0$. If (iv) holds, then $n = i+k$. In this case, we show that $P_{i,j}(2) \neq$

$P_{k,l}(2)$ to prove that the third CE's are not common. $P_{k,l}(2) = \langle k+l \rangle_n = \langle j \rangle_n \neq P_{i,j}(2) = \langle i+j \rangle_n$. Suppose now that (ii) is true. If (iii) holds, then $n = i+k$ and by the above argument, the third CE's are not common. If (iv) holds, then $i+k = 2n$, but $i, k \leq \alpha = n-2$, so this cannot occur. Therefore, if $j, k \neq n$, then the paths do not have a common third CE.

Now let $j = n$. In this case, $\langle k+l \rangle_n = 0$ and $\langle l \rangle_n = \langle i \rangle_n$, so $k+i = n$. For other values of i and k, no equal *2-prefix* is possible. For such values of i and k, $P_{i,j}(2) = \langle i \rangle_n \neq P_{k,l}(2) = \langle k+i \rangle_n$, so the paths do not have a common third CE if $j = n$. The case for $k = n$ is similar.

To prove the remaining cases, we will be trying to establish conditions under which two sets X and Y are equal, in which the sets have the form:

$$X = \{\langle x_1 \rangle_n, \langle x_2 \rangle_n, \langle x_2+1 \rangle_n, \langle x_2+2 \rangle_n, \cdots, \langle x_2+m-2 \rangle_n\},$$

$$Y = \{\langle y_1 \rangle_n, \langle y_2 \rangle_n, \langle y_2+1 \rangle_n, \langle y_2+2 \rangle_n, \cdots, \langle y_2+m-2 \rangle_n\},$$

$\langle x_1 \rangle_n \neq \langle y_1 \rangle_n$, and $|X| = |Y| \leq 3$.

Lemma B.3. If $X = Y$, then either (i) $\langle x_1 \rangle_n = \langle x_2-1 \rangle_n$ and $\langle y_1 \rangle_n = \langle y_2+m-1 \rangle_n$ or (ii) $\langle x_1 \rangle_n = \langle x_2+m-1 \rangle_n$ and $\langle y_1 \rangle_n = \langle y_2-1 \rangle_n$ must be true.

Case 3: $3 \leq m \leq n-k$. For this range of m, the *m-prefix* sets are as follows.

m-prefix set for $P_{i,j}$: $\{j-1, \langle i+j-1 \rangle_n, \cdots, \langle i+j+m-3 \rangle_n\}$

m-prefix set for $P_{k,l}$: $\{l-1, \langle k+l-1 \rangle_n, \cdots, \langle k+l+m-3 \rangle_n\}$

For these two sets to be equal, then by Lemma B.2, either (i) $\langle j \rangle_n = \langle i+j-1 \rangle_n$ and $\langle l-2 \rangle_n = \langle k+l+m-3 \rangle_n$ or (ii) $\langle l \rangle_n = \langle k+l-1 \rangle_n$ and $\langle j-2 \rangle_n = \langle i+j+m-3 \rangle_n$. For (i), if $\langle l-2 \rangle_n = \langle k+l+m-3 \rangle_n$, then $\langle k+m-1 \rangle_n = 0$, so $m = n-k+1$, but this is not possible by the range on m. For (ii), if $\langle j-2 \rangle_n = \langle i+j+m-3 \rangle_n$, then $\langle i+m-1 \rangle_n = 0$, so $m = n-i+1$, but this is not possible by the

range on m. By Lemma B.2, no m exists such that $2 \le m \le n-k$ and the *m-prefix* sets are equal.

Case 4: $n-k+1 \le m \le n-i$. For this range of m, the *m-prefix* sets are as follows.

$$m\text{-prefix set for } P_{i,j}\text{: } \{j-1, <i+j-1>_n, \cdots, <i+j+m-3>_n\}$$

$$m\text{-prefix set for } P_{k,l}\text{: } \{l-1, <l-m-1>_n, \cdots, <l-3>_n\}$$

For these two sets to be equal, then by Lemma B.2, either (i) $<j>_n = <i+j-2>_n$ and $<l-2>_n = <l-3>_n$ or (ii) $<l>_n = <l-m-1>_n$ and $<j-2>_n = <i+j+m-3>_n$. For (i), clearly $<l-2>_n \neq <l-3>_n$, so (i) cannot hold. For (ii), if $<j-2>_n = <i+j+m-3>_n$, then $<i+m-1>_n = 0$ and $m = n-i+1$, but this is not possible by the range on m. By Lemma B.3, no m exists such that $n-k+1 \le m \le n-i$ and the *m-prefix* sets are equal.

Case 5: $n-i+1 \le m \le n-1$. For this range of m, the *m-prefix* sets are as follows.

$$m\text{-prefix set for } P_{i,j}\text{: } \{j-1, <j-m-1>_n, \cdots, <j-3>_n\}$$

$$m\text{-prefix set for } P_{k,l}\text{: } \{l-1, <l-m-1>_n, \cdots, <l-3>_n\}$$

For these two sets to be equal, then by Lemma B.2, either (i) $<j>_n = <j-m-1>_n$ and $<l-2>_n = <l-3>_n$ or (ii) $<l>_n = <l-m-1>_n$ and $<j-2>_n = <j-3>_n$. For (i), clearly $<l-2>_n \neq <l-3>_n$, and for (ii), clearly $<j-2>_n \neq <j-3>_n$. By Lemma B.3, no m exists such that $n-i+1 \le m \le n-1$ and the *m-prefix* sets are equal.

Combining the above arguments, we find that paths $P_{i,j}$ and $P_{k,l}$ are CE disjoint for $j \neq l$. □

Proof for Lemma 4.1 :

The primary differences between Lemma 4.2 and Lemma 4.3 are that in the former case, $\alpha = \left\lceil \frac{n}{2} - 1 \right\rceil$ and the properties relate to both PE's and CE's, while in

the latter case, $\alpha = n-2$ and the properties relate only to CE's. Consequently, the proofs for the properties for Lemma 4.1 will be based on those for Lemma 4.2.

Property N1. Paths $P_{i,j}$ and $P_{i,l}$ are PE and CE disjoint, for $j \neq l$.

Proof: *Property N1* is implied by *Property N3*, proved below, for $i = k$. □

Property N2. Paths $P_{i,j}$ and $P_{k,j}$ have $(i+1)$ common PE's and CE's, for $i < k$.

Proof: By *Property L2* of Lemma 4.2, the paths have $(i+1)$ common CE's. Observe from the proof of *Property L2* of Lemma 4.2 above that the paths have exactly $i+1$ values of m for which equal *m-prefix* sets exist. By Lemma B.1, the paths also have $(i+1)$ common PE's. □

Property N3. Paths $P_{i,j}$ and $P_{k,l}$ are PE and CE disjoint, for $j \neq l$.

Proof: The proof for *Property L3* of Lemma 4.2 above was partitioned into five cases. In Cases 1, 3, 4, and 5, we proved that no m in the given ranges exists such that the paths have equal *m-prefix* sets. Since α for Lemma 4.1 is smaller than α for Lemma 4.2, this fact is also true for Lemma 4.1. It was only in Case 2 ($m = 2$) that it was possible for any equal *2-prefix* sets to exist. We establish that no equal *2-prefix* sets exist for $\alpha = \lceil n/2 - 1 \rceil$. In Case 2, the only circumstances in which equal *2-prefix* sets were possible were when $i+k = n$. For Lemma 4.1, $i, k \leq \alpha = \lceil n/2 - 1 \rceil$, so no values of i and k exist for which $i+k = n$. Combining all cases, no value of m exists for which $P_{i,j}$ and $P_{k,l}$ have equal *m-prefix* sets, so by Lemmas B.1 and B.2, the paths are PE and CE disjoint. □

The following three properties apply to group $G'_{\alpha+1}$ for cases in which n is even. Observe that $\alpha+1 = n/2$.

Property N4. Paths $P_{n/2,j}$ and $P_{n/2,l}$ are PE and CE disjoint, for $j \neq l$.

Proof: *Property N4* is implied by *Property N6*, proved below, for $i = n/2$. □

Property N5. Paths $P_{i,j}$ and $P_{n/2,j}$ have $(i+1)$ common PE's and CE's, for $i < n/2$.

Proof: Same as proof for *Property N2*. □

Property N6. Paths $P_{i,j}$ and $P_{n/2,l}$ are PE and CE disjoint, for $j \neq l$.

Proof: As for *Property N3*, this proof largely follows the proof of *Property L3*. Consider the proof of *Property L3* with $k = n/2$. In Cases 1, 3, 4, and 5 of the proof, no m in the given ranges exists such that the paths have equal *m-prefix* sets. Since α for Lemma 4.1 is smaller than α for Lemma 4.2, this fact is also true for Lemma 4.1. It was only in Case 2 ($m = 2$) that it was possible for any equal *2-prefix* sets to exist.

In Case 2, the only circumstances in which equal *2-prefix* sets were possible were when $i+k = n$. For *Property N6*, this can occur when $i = k = x = n/2$. Consider paths $P_{n/2,j}$ and $P_{n/2,l}$, where $1 \leq j, l \leq n/2$ and $j \neq l$. As described in the proof of *Property L3*, for the *2-prefix* sets to be equal, one of (i) $j = n/2 + l$ or (ii) $j = n/2 + l - n = l - n/2$ and one of (iii) $l = n/2 + j$ or (iv) $l = n/2 + j - n = j - n/2$ must be true. Since $1 \leq j, l \leq n/2$, none of (i), (ii), (iii), or (iv) can be true, so equal *2-prefix* sets are not possible.

Combining all cases, no value of m exists for which $P_{i,j}$ and $P_{k,l}$ have equal *m-prefix* sets, so by Lemmas B.1 and B.2, the paths are PE and CE disjoint. □

Vita

Mr. Soh received a B.S. in Electrical Engineering (Computer option) from the University of Wisconsin, Madison, in 1986. Since 1987, he has been a graduate student at Louisiana State University, where he received his M.Sc. in Electrical & Computer Engineering in 1988. He is pursuing a Ph.D. in Electrical & Computer Engineering. His research interests include computer architecture, fault tolerant computing, parallel processing, and reliability of digital systems. Mr. Soh has several publications which have appeared in *IEEE Trans. Parallel and Distributed Systems, IEEE Trans. Reliability, International Journal on Electrical and Computer Engineering*, and in International conferences. Mr. Soh is a student member of IEEE, Eta Kappa Nu, and Tau Beta Pi.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Sieteng Soh

Major Field: Electrical Engineering

Title of Dissertation: Reliability Analysis of the Hypercube Architecture

Approved:

Suresh Rai

Major Professor and Chairman

David Boyd

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

2/9/1993