This is to certify that the

dissertation entitled

On Some Topological Issues In
Message-Passing
Multiprocessor Architectures

presented by

Guy Warren Zimmerman

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

A. H. Esfahanian
Major professor

Date 7-13-90

ON SOME TOPOLOGICAL ISSUES IN MESSAGE-PASSING MULTIPROCESSOR ARCHITECTURES

By

Guy Warren Zimmerman

A DISSERTATION

Submitted to
Michigan State University
in partial fullfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1990

ABSTRACT

ON SOME TOPOLOGICAL ISSUES IN MESSAGE-PASSING MULTIPROCESSOR ARCHITECTURES

By

Guy Warren Zimmerman

Advances in computer technology have made it possible to construct machines with very large numbers of processors. Among such machines are the distributed-memory machines, termed Multicomputers, which have the potential to deliver very high performance for a large class of applications. A major component in the design of a Multicomputer system is its underlying interconnection topology (usually modeled by a graph), since it has a significant influence on virtually every other aspect of the system, such as communication capabilities and reliability.

This dissertation considers some aspects of communication and reliability in a Multicomputer system which are related to the underlying topology of the system. In particular, a type of communication known as broadcasting is studied. The roles of trees in completing broadcasting, and a near optimal broadcasting algorithm for a specific Multicomputer system known as the De Bruijn networks are explained. Other partial results such as the distribution of the broadcast times of trees of different orders are given.

The reliability, or fault-tolerance, issues discussed here are of topological nature. Specifically, a new approach to system-level fault-tolerance in Multicomputers is presented. In this approach a bound is placed on the maximum number of connections allowed at each processor and the number and configuration of redundant components is then determined to achieve a specified level of fault-tolerance. The approach is applied to the design of fault-tolerant ring topologies. Designs are presented which can tolerate up to three processor failures.

To my wife, Janet

ACKNOWLEDGMENTS

I would like to thank my advisor, Abdol-Hossein Esfahanian for all of his assistance during my Ph.D studies, for his well chosen words of encouragement and for his confidence in me. I would also like to thank Dr. Lionel Ni, Dr. Anthony Wojcik, and Dr. Bruce Sagan for their service on my committee and for their helpful comments and suggestions regarding the preparation of this dissertation. I would like to thank my wife, Janet Ergo Zimmerman, without whose support and sacrifice I could not have completed my Doctoral work. I would like to thank my parents, Jack and Jane Zimmerman, for instilling in me the importance of learning and for giving me the opportunity to pursue a college education. Finally, I wish to thank Jack and Carol Ergo and my grandparents, Mable and Harry Raymond, and for their years of support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Advances in computer technology have made it possible to construct machines with very large numbers of processors. Such machines have the potential to deliver very high performance for a large class of applications. In fact, the development of multiple-processor computers has made it possible to sharply reduce the amount of time required to solve large scale problems and as a consequence, to greatly increase the scale of problems which can be solved in a reasonable amount of time. These problems are most often of a scientific nature, are very CPU intensive, and typically require large amounts of memory. Some examples of such problems include: aerodynamic simulations, weather forecasting, petroleum exploration, image processing, and artificial intelligence [HwBr84].

1.1. Multiprocessors and Multicomputers

At the highest level, the major components of the architecture of a multiprocessor system are the processors, memory unit(s) and the interconnection network through which the processors communicate with each other and access the memory unit(s).

Interconnection networks can be classified in two categories: *static networks* and *dynamic networks* [Feng81]. The difference is that the physical connections may change in a dynamic network, while they always remain fixed in a static network. Examples of static networks include the ring, tree, 2-D mesh and hypercube. Dynamic networks include *single-stage* and *multistage* structures. Single-stage is also called *recirculating network*, which is composed of a stage of switch boxes cascaded to a link connection pattern. A typical multistage interconnection network in a multiprocessor system with N processors consists of $\log_k N$ stages with N/k $k \times k$ switch boxes at each stage, where $2 \leq k \leq 8$. A multistage network can usually connect an arbitrary input to an arbitrary output. Examples of multistage networks include Omega (shuffle-exchange) and Delta networks. Dynamic networks offer greater flexibility at the cost of additional hardware and the overhead of managing the interconnection network [Agra83, BhAg83].

Over the years, two broad categories of multiple-processor computer systems have emerged, based largely on the degree of autonomy of the processors and the way the memory is organized and controlled. The first of these classes of systems is referred to as *SIMD*, an acronym for Single Instruction Multiple Data. SIMD machines are characterized by the fact that at any given instant, every enabled processor in the system is executing the same instruction as all the other processors, although each may be processing different data. For many applications with inherent massive data parallelism, SIMD machines can provide high performance at a reasonable cost. Several successful SIMD machines have been developed, including MPP, ICL's DAP, and the Connection Machine [Flan77, Batc80, Hill85]. One of the best features of these machines is their ease of programming. This feature is also the principal drawback in that the ease of programming is made possible by the limited range of applications which can be effectively executed.

The other major category of multiple-processor machines is the *MIMD* class; MIMD being an acronym for Multiple Instruction Multiple Data. As the name indicates,

at any given time each processor in the system may be executing a different instruction, and is typically processing different data. These machines are a more general class of multiple-processor computer systems, since they may simulate the operation of SIMD machines. MIMD machines are further categorized by the way the memory is organized and managed. A *shared-memory* machine has a single global memory which is usually equally accessible to all its processors. Since having a single memory can lead to memory contention problems, the memory is often physically organized into modules and the processors access the modules through a communication network, most frequently multistage. Even so, the possibility for contention for access to specific modules can still be a problem. Processors communicate with one another by accessing prearranged locations in the shared-memory. Examples of shared memory machines include the BBN Butterfly, the NYU Ultracomputer, the IBM RP3 , the Sequent Balance/Symmetry, the Encore Multimax and the Cray X-MP/Y-MP [Gott83, Lars84, Crow85, Pfis85, Sten88, RCCT90]. The chief advantage of shared memory machines is that they allow the programmer the impression of a single address space. The major disadvantages are that the machines do not scale well and contention for memory access can lead to so-called "hot-spots" which can degrade performance.

The second type of MIMD machine is referred to as a *distributed-memory* machine. Here each processor in the system has its own local memory (which it controls) and the processors are connected by a communication network (most frequently static). The programmer does not have the advantages of a single global address space. Processors communicate by passing messages through the communication network, and a message may have to go through several intermediate processors before reaching its destination. For this reason, these systems have also been referred to as *message-passing* or *point-to-point* systems. A number of research and development projects have been undertaken to construct such machines [Seit85, HMSC86, SAFM88]. Among the existing systems, the most dominant one is the hypercube which is commercially available in different

variations including the Mark III, the iPSC-1, and the Ncube [PTLP85, GuHS86, GrRe86]. In contrast to shared-memory machines, message passing architectures generally scale better and are more robust. However, they are more difficult to program effectively; and since communication is usually slower, they are better suited to problems where the communication/computation ratio is relatively small. This dissertation concerns several problems related to this type of machine, which we will refer to as a *Multicomputer (MC).*

1.2. Motivation and Problem Statement

There are many aspects to the design of an MC. As in the development of every computer there is the broad hardware/software dichotomy; and within each of these two areas, one may consider many different levels of design. Here we will be concerned with *system-level* issues. System-level means the consideration of the MC in a macroscopic fashion. In terms of software, the issues are the development of operating systems, compilers, programming tools, etc., all of which must address the MC as a complete entity. In the hardware area, system-level means viewing the MC as a collection of processor/memory-pairs connected by communication links. These two areas are highly interdependent, and a successful design must consider both areas as they relate to each other and to the design goals.

In terms of hardware, at the system-level the selection of the topology of the interconnection network is fundamental since it has a significant influence on virtually every other aspect of the MC including communication capabilities and system reliability. In a certain sense, the topology *is* the MC. Communication characteristics such as *message delay* and *routing* are directly related to the underlying topology [FaKr83]. Many research articles have focused on the relationship between the topology and one or more of these characteristics. As a result, many topologies for MCs have been proposed including cube-connected cycles, binary tree, 2D-mesh, binary DeBruijn networks, and

k-ary n-cube [Prad81, PrVu81, HsYZ87, HwGh87, DuHw88, ChCD88, Dall90]. This dissertation will examine how the topology of an MC influences its communication capabilities and reliability.

Since MCs rely on passing messages, the performance of an MC is significantly tied to the types of communication that can be accomplished, and how quickly and efficiently they can be done. Experience has shown that several communication paradigms are useful in developing algorithms for MCs. These paradigms include: *one-to-one* where a processor wishes to send a message to another specific processor; *one-to-all* where a processor wishes to send a message to all the other processors in the system; and finally *one-to-many*, which is a generalization of the previous two. These classes have also been referred to as *unicast*, *broadcast* and *multicast*, respectively. Some of the issues in implementing these communication paradigms are: the number of messages that a processor may simultaneously send or receive, the type of routing strategy (e.g., static or dynamic), and the type of switching strategy (e.g. packet switched, circuit switched, wormhole, virtual cut/through) [KeKl79, DaSe87]. The performance of each of these communication schemes is affected by the topology. In this dissertation we consider the problem of broadcasting in MCs and we assume that each processor may send or receive only one message at a time.

The large number of components (such as processors and memories) and the overall complexity of such systems serve to exacerbate reliability problems, and much research has been done in studying reliability issues in MCs [KuRe80, PrRe82]. Several types of reliability have been distinguished based principally on the nature of the applications the MC is to execute. A *highly available system* may have frequent failures, but the failures are such that the system can be restored to proper functionality in a short time; these systems are "up" most of the time. The *Electronic Switching System* (ESS) used by AT&T is designed for high availability [Prad86]. An *ultra reliable* system is one which has very few failures over a specified time interval. This type of reliability is demanded by

applications where failures can have catastrophic results. Examples of such applications include operation of aircraft and medical monitoring equipment.

One approach to developing reliable systems is to use ultra-reliable components in the construction. However, this can be an expensive proposition and does not always adequately address the problem. Another alternative is the so-called *fault avoidance* approach in which the system is designed to avoid faults from occurring. The approach we will consider in this dissertation is referred to as *fault-tolerance*. As the name implies, the idea is to design systems in which faults may be tolerated, and a system will be said to be fault-tolerant if it can remain *functional* in the presence of failures. At the system level, two types of failures are considered: the failure of a set of processors, and the failure of a set of communication links. What constitutes a functional system depends largely on the type of applications that the system will execute.

Two basic functionality criterion have received much attention. According to one of these, an MC is considered to be functional as long as there is a non-faulty communication path between all pairs of non-faulty processors. In graph theoretic terms, the underlying topology is *connected*. This type of functionality is applicable to the so-called coarse grain application, in which the algorithms are relatively insensitive to topology. Typically, each processor is made responsible for executing some subset of the tasks required to solve some (large) problem; in the event of a failure, the tasks are assigned to other processors. This is the so called fault-tolerance through degraded performance approach [ArLe81, Esfa88, RaGA85].

The second functionality criterion considers an MC to be functional only when a desired topology is contained in the system. In the past few years, much work has been done in developing efficient, high performance parallel algorithms for various MCs. For many of these algorithms, the existence of certain topologies is a significant factor in delivering the desired performance. For such applications, the system should be able to provide a specific topology throughout the execution of the algorithm. This criterion was

first formulated by Hayes, and subsequently a number of fault-tolerant (in the sense of the second criterion) topologies have been proposed [Haye76]. The basic approach in this case is to employ redundant components (links or processors) throughout the system. In the event of a failure, the system could then be *reconfigured* to exclude the faulty component by using one of the redundant ones. In this dissertation, fault tolerance will refer to the second functionality criterion, and we will consider only processor failures.

Broadcasting and fault-tolerance have much in common in that they are both intimately tied to the topology of the MC. In addition, broadcasting can be an integral part of the reconfiguration process for a fault-tolerant system. Also, a broadcasting scheme for a fault-tolerant system must be robust enough to accommodate faulty system components. A number of papers have addressed problems common to these two areas [Bien88, LeHa88, Lies88].

1.3. Thesis Organization

As indicated, this dissertation will focus on two problems both of which are related to the topology of the interconnection network. Such a topology is conveniently modeled as a graph where the nodes of the graph represent the processors and the edges represent the communication links. Many problems in Computer Science and Engineering can be so formulated. However, it is often the case that such models are only practical for "small" instances of problems, since even modestly sized graphs can become incredibly complicated and unwieldy to work with. In addition, many of the operations one wishes to perform on such models are of a trial and error genre, and such operations become increasingly labor-intensive for all but the smallest graphs. To address this problem, we have developed a software package for graph manipulation which has been instrumental in obtaining some of the results in this dissertation and other research as well. This package is described in Chapter 2, where we also present graph theoretic definitions and terminology which will be used throughout the dissertation.

Broadcasting in MCs is discussed in Chapter 3. In particular, we examine the role that *trees* play in the broadcasting process. A partial characterization of a class of graphs called *minimum broadcast trees* is given. Finally, we present a distributed algorithm for doing broadcasting in networks based on Binary DeBruijn graphs.

Fault-tolerance in MCs is the subject of Chapter 4. In this chapter we introduce a new approach to system-level fault-tolerance. The approach is applied to the case when the topology of the MC is a ring. For this case, a class of graphs called *Chordal Rings* is shown to be an optimal solution.

In the final chapter we present our conclusions, summarize our research contributions and suggest some directions for future study.

Chapter 2

Graphs and GMP

We will use graphs to model the topology of an MC. The vertices of the graph represent processors and the edges of the graph represent the communication links. With this model in mind, we will use the terms vertex, node, and processor interchangeably and similarly for the terms edge and link. In this chapter, we present our graph definitions, notation and terminology which are common to both of the two problem areas that we consider in subsequent chapters. Definitions and terms which are specific to a problem area will be introduced in the context in which they are needed. We also describe a software graph manipulation package which we have developed and which has been useful in our research.

2.1. Graph Definitions, Notation, and Terminology

In this section we present our graph theoretic definitions and notation. Graph theoretic terms not defined here can be found in [Hara72]. Let $G(V,E)$ be a finite graph without loops or multiple edges with the vertex set $V = V(G)$ and the edge set $E = E(G)$. The *order (size)* of $G(V,E)$ is equal to the cardinality of the vertex set (edge

set), and is denoted by $|V(G)|$ ($|E(G)|$) or simply $|V|$ ($|E|$) when the context is clear. If an edge $e = (u,v) \in E$, then vertices u and v are said to be *adjacent*, and the edge e is said to be *incident* to these vertices. For a vertex $v \in V$, $I(G:v)$ represents the set of all edges incident to v in G. Two vertices connected by an edge will be referred to as *neighbors*. The degree of a vertex v is $d(v) = |I(v)|$. The *degree sequence* of G is a non-decreasing list of the degrees of all the vertices in G. The *minimum* and *maximum* degrees of a graph G are $\delta(G) = \min \{ d(v) \mid v \in V \}$ and $\Delta(G) = \max \{ d(v) \mid v \in V \}$, respectively. A graph G is *r-regular* if for all $v \in V$, $d(v) = r$. A graph is a *cubic* graph if it is 3-regular.

A graph H which has all of its vertices and edges in G is a *subgraph* of G, denoted $H \subseteq G$. If G is isomorphic to a subgraph of H, we say G may be *embedded* in H. If H is a subgraph of G, then G is a *supergraph* of H. A *spanning subgraph* of G is a subgraph containing all the vertices in G. For a set $F \subseteq V(G)$, the notation $G-F$ represents the subgraph of G obtained by removing from G all the vertices in F along with their incident edges.

A *path* P is a sequence of distinct vertices $v_0,v_1,...,v_{n-1},v_n$ where $(v_i,v_{i+1}) \in E(G)$, $0 \le i \le n-1$. The *length* of the path is the number of edges in the path. A graph is *connected* if every pair of vertices are joined by a path. A *cycle* is a sequence of vertices $v_0,v_1,...,v_{n-1},v_n$ where $(v_i,v_{i+1}) \in E(G)$, $0 \le i \le n-1$, $n \ge 3$, $v_0 = v_n$ and all the other vertices are distinct. The cycle of order N, also called an N-cycle, will be denoted as C_N. When convenient, a cycle will be specified by listing, in order, the vertices in the cycle. For example, x_0, x_1, x_2, x_3, x_0 defines a 4-cycle. A *Hamiltonian* graph contains a cycle passing through all its vertices.

The *distance* between two vertices u and v, $dist(u,v)$, is defined as the length of a shortest path joining these vertices. The *diameter* of graph G, $D(G)$, is the largest value of $dist(u,v)$ in G. A connected acyclic graph is also called a *tree*. The vertices of degree one in a tree, $T(V,E)$, are called the *leaves* of T. A *rooted* tree is one in which a specific

node is labeled as the root. When no such designation is made, the tree is said to be *free*. The *level* of a node in a rooted tree is the length of the unique path from the root to the node. The *height* of the tree is the number of levels or the length of the longest path from the root to any node in the tree. A spanning subgraph which is also a tree is called a *spanning tree*. G is a *bipartite* graph if $V(G)$ can be partitioned into two disjoint subsets V_1 and V_2 such that every edge of G joins a vertex $u \in V_1$ with a vertex $v \in V_2$. In such a case G may be denoted $G(V_1,V_2,E)$.

2.2. GMP - A Software Package for Graph Manipulation

In this section we give a brief description of the software package for graph manipulation that we developed. A more detailed description, including a user's manual, programmers manual, as well as the source code can be found in [ZiEs88]. We will describe the functionality of the package, its utilization to date and some suggestions for improvement.

Graph theory has long become recognized as one of the more useful research tools in Computer Science and Engineering. Extensive use of graph theory has been made in areas such as topological design of networks, complexity theory, and design and analysis of algorithms. This dissertation considers two such problems.

In the course of our work in graph theory, we had increasingly noticed the need for a software package which would enable the user to interactively manipulate graphs, test conjectures, etc. No such package was then available, partially due to the unavailability of affordable graphics workstations. In the summer of 1986, when such workstations became accessible to us, we undertook a project to develop such a software package. We dubbed it GMP, an acronym for Graph Manipulation Package. The package was written in C and currently runs in the SunView environment on SUN workstations.

The first version of GMP became available in the winter of 1987. It is an interactive program which allows users to visually manipulate graphs with up to 100 vertices. There

are essentially two major facets of the program. First, GMP allows the user to construct and modify graphs on the screen. This construction can be done manually using a mouse. Further, a menu of standard graphs is available, or the user can create a graph *off-line* and then load it into GMP. Second, GMP allows the user to investigate the interaction of graph algorithms and graphs. A number of *classic* algorithms are built into GMP, and the program has been designed to allow users to easily incorporate new algorithms into the system. Facilities exist for storage and retrieval of graphs, as well as for obtaining publication quality hardcopies of created graphs.

In developing GMP, our initial intent was to produce an exploratory tool rather than a results-oriented, computational program ala LINPACK, etc. We wanted something which would allow us to easily test conjectures and investigate ideas. This orientation had important consequences on the overall design. One such consequence was the decision to limit the maximum size of graphs to 100 vertices. While the specific number was somewhat arbitrary, it seemed that any number much higher would have decreased the user's ability to get a good understanding of what was going on. Also, our experience was that any size monitor gets crowded with more than 100 vertices. Our design goal also affected various aspects of the implementation. For example, data structures were selected for their simplicity and universality, rather than economy of memory, etc. Further, in adding an algorithm to the package, we generally chose a version which was easier to implement, rather than one with the best performance. This was in keeping with the above philosophy, and it was thought that for "small" graphs, the decreased performance was probably minimal.

2.2.1. An overview of GMP

The GMP user interface consists of three main windows: the message window, the control window, and the canvas window (Figure 2-1). Virtually all user input to GMP is done via the mouse, although the keyboard is required for some functions. The message

window is the long horizontal window at the top of the screen. One purpose of this window is to provide feedback and instructions to users. This window also contains five pulldown menus which are accessed by clicking the RIGHT mouse button over the menu title (Figures 2-2 - 2-5). Each of these menus is described in more detail later. The long vertical window on the right is the **control window**. From this window the user can initiate file system commands to store and retrieve graphs and select which manipulation command is to be active when the cursor is in the canvas window. The large square window is the **canvas window**. This is the window in which the graph is drawn and manipulated. Most user feedback occurs through this window.

As stated earlier, GMP has two major overlapping functions: 1) creating and manipulating graphs, and 2) invoking algorithms which act on the created graphs. Graphs can be created in several ways. Some standard graph definitions are available in the **Std Graphs** pulldown menu in the message panel. Selecting one of these options will create a graph of the specified type using the default number of vertices. In Figure 2-2, the Std. Graphs menu is shown with the Binary Tree option highlighted. The graph which is generated by this command is shown in the canvas window. Note also the **default parameters** window, in which the parameter *number of vertices* is set at 20. This indicates the order of the graph that will be created by the Std. Graphs Menu. Graphs can be created manually using one of the manipulation commands selected from the control window. For example, select *Add Vertex*, move the cursor into the canvas and click where you would like a new vertex to be placed. In addition, you can type an adjacency matrix into a file, add an appropriate GMP header and then load this graph into GMP. The vertices and edges of graphs created using one of these three methods all have default values for their parameters: edge weight, node weight, edge type, node type, etc. The default values for each may be set by the user, and the user may also manually set the value of any parameter for a particular vertex and/or edge.

The **Miscellaneous** menu (see Figure 2-3), as the name implies, contains a number of commands for various purposes. Among these are: displaying/hiding graph labels, manipulating the weights associated with the graph vertices/edges, and altering the lay-out of the graph. Figure 2-3 shows the main miscellaneous menu along with the *Weight Options* submenu. The *Show Edge Weights* option is highlighted and the results of this command are seen in the canvas: all edges in the graph shown have weight equal to 1.0.

Users invoke algorithms on created graphs via the **Algorithms** menu (see Figure 2-4). The *Shortest Path* algorithm is highlighted and the results of the command are displayed in the canvas. In this example, a shortest path from a source vertex (blackened) to all the other vertices in the graph is shown by the "double" edges. In addition, the distance from the source to each vertex is displayed next to the vertex. The figure also shows an example of another graph type available in the Std. Graphs menu: 2D-Mesh. A second example of the results of an algorithm is seen in Figure 2-5. The *Find Cycles* algorithm was invoked and the user requested a search for a cycle of order 26. Such a cycle was found and displayed in the canvas using double edges.

As mentioned, the package was designed to allow users to easily add new algorithms. A programmer's manual is available containing programming guidelines, examples and a description of user-accessible utility procedures. User implemented algorithms are accessed through the **User Algs** menu. This menu is user-definable and may contain an unlimited number of user algorithms.

Finally, the **Goodies** menu (Figure 2-6) allows the user to perform a variety of operations which alter the visual appearance of the graph.

2.2.2. Utilization of GMP

Since its inception, GMP has been used in conjunction with graph theory related courses in both the Mathematics and Computer Science departments at Michigan State University. In particular, in the Computer Science course entitled *Analysis of Graph*

Algorithms students have used the package in a *discovery* mode to investigate basic concepts in graph theory. The interactive and visual features have been extremely motivating in this context. The package also provides a framework from which one can develop and test new graph algorithms, as well as to study existing ones. Students have written and/or coded algorithms as projects for this class.

In addition to its instructional use, GMP has also been a valuable research tool. Briefly, it automates many of the mundane operations that a graph theorist would normally do by hand. This automation allows many conjectures to be easily tested and refined, a task which is often too onerous to be done manually. The package has been instrumental in obtaining results reflected in recent publications. In particular, GMP has expedited the formulation of a new approach to system-level fault-tolerance and communication paradigms for multicomputers [EsNS89, ZiEs90]. In addition, some of the proofs in Chapter 4 of this dissertation would have not been possible without it. It also served as a tool for producing all the graph related graphics in this dissertation and other recent publications.

The package has also been made available to researchers at other institutions including Western Michigan University, Grand Valley State University, George Washington University, Georgia Institute of Technology, Rutgers University, Boston University, University of Iowa, and Université Paris-Sud. The responses have all been positive. To date, GMP is unique in its category of software.

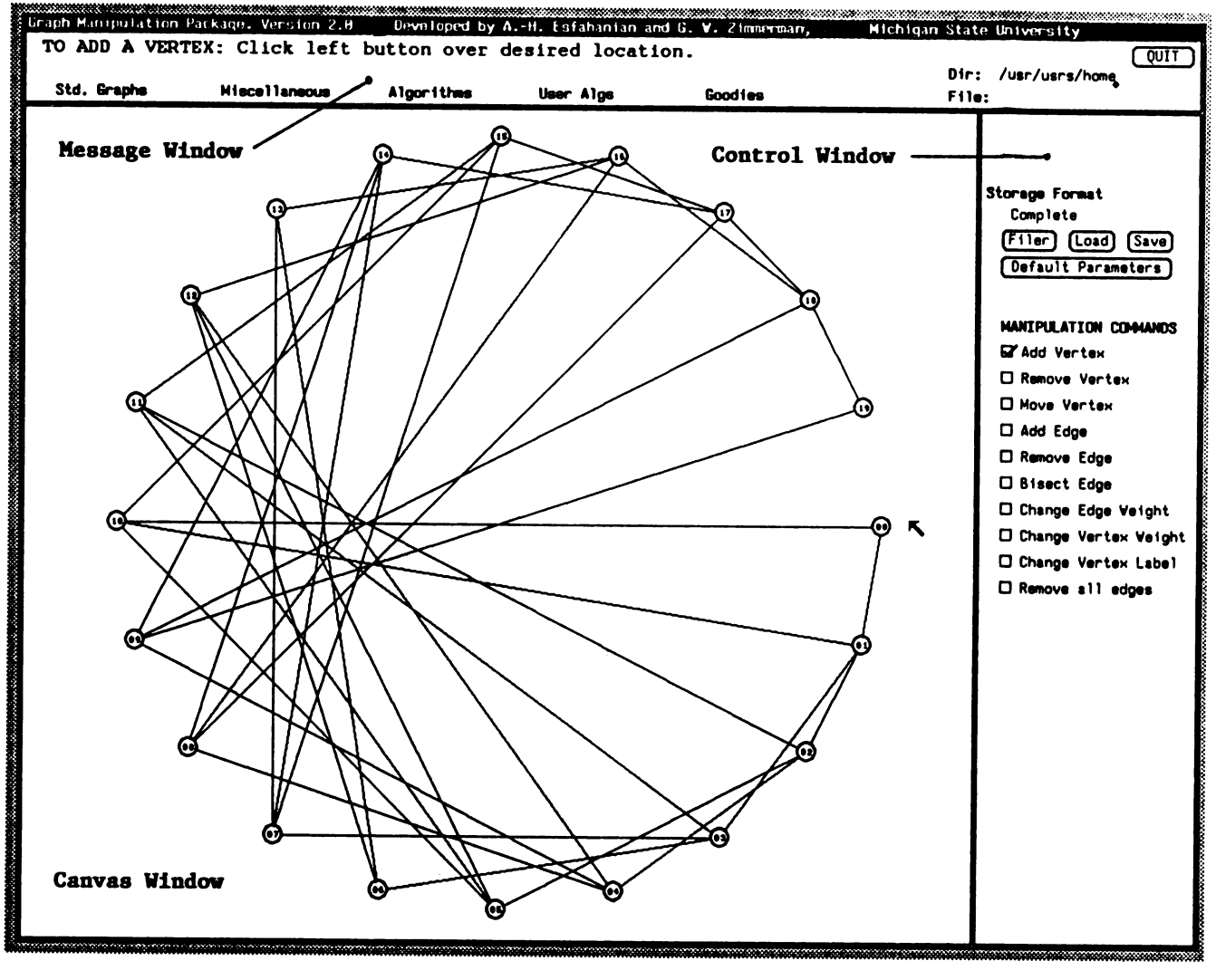

Figure 2-1. The three principal GMP windows.

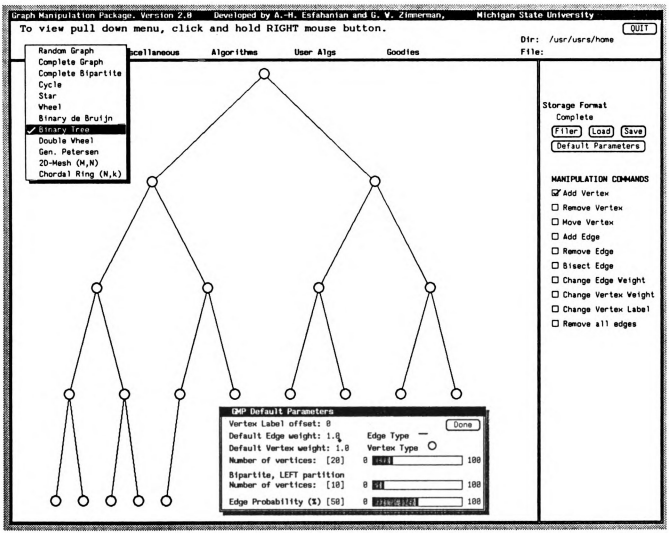

Figure 2-2. The Std. Graphs menu.

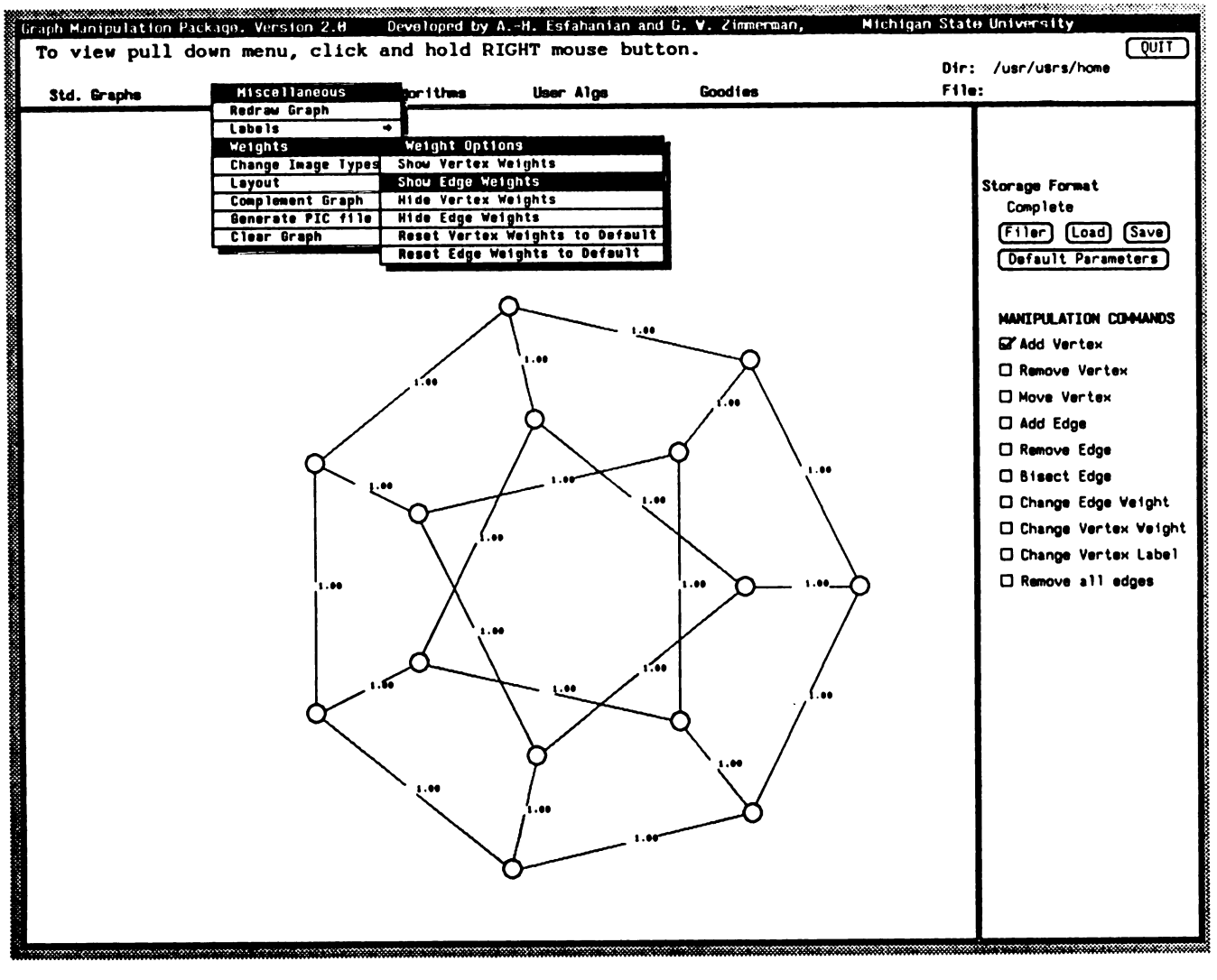

Figure 2-3. The Miscellaneous menu.

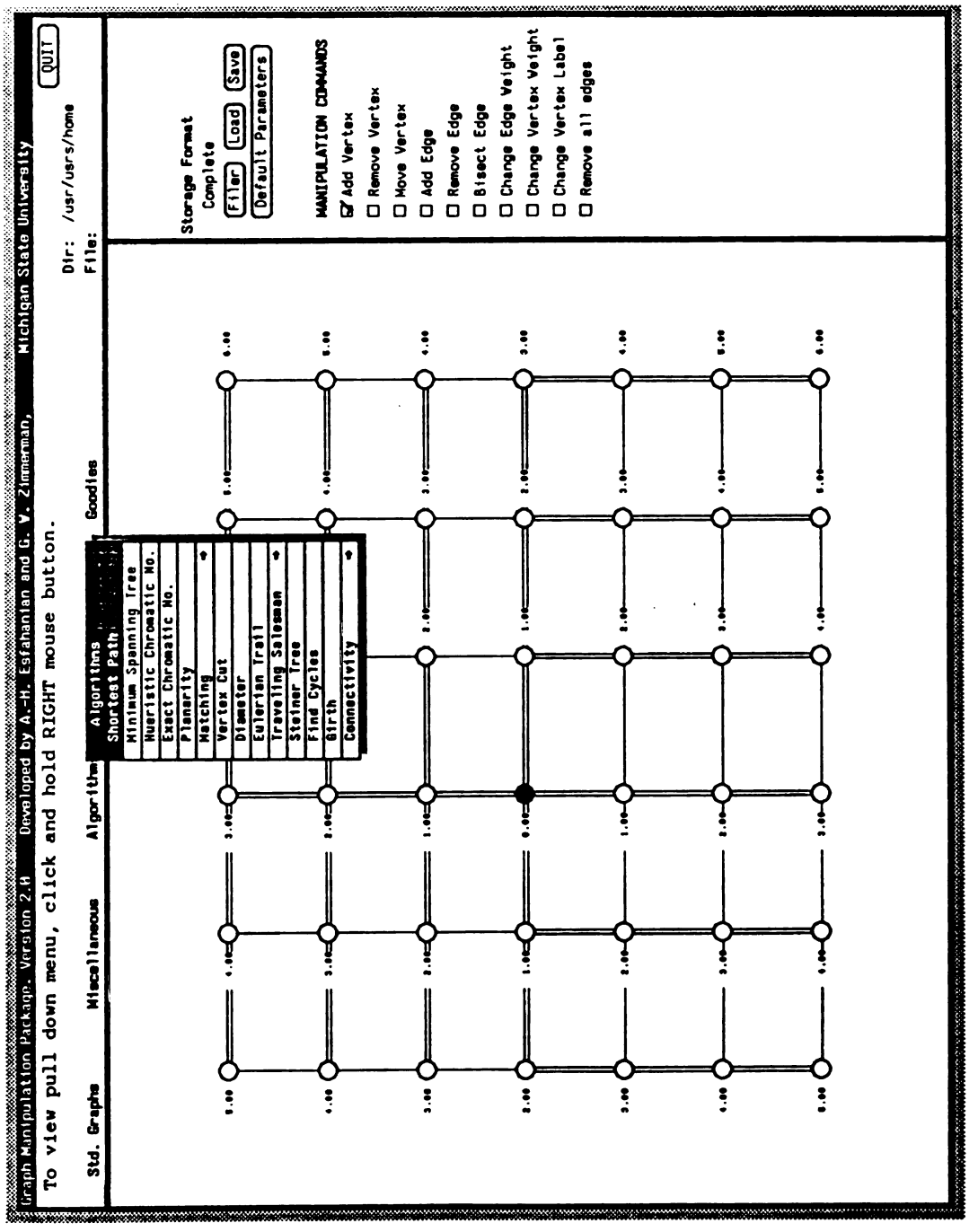

Figure 2-4. The Algorithms menu.

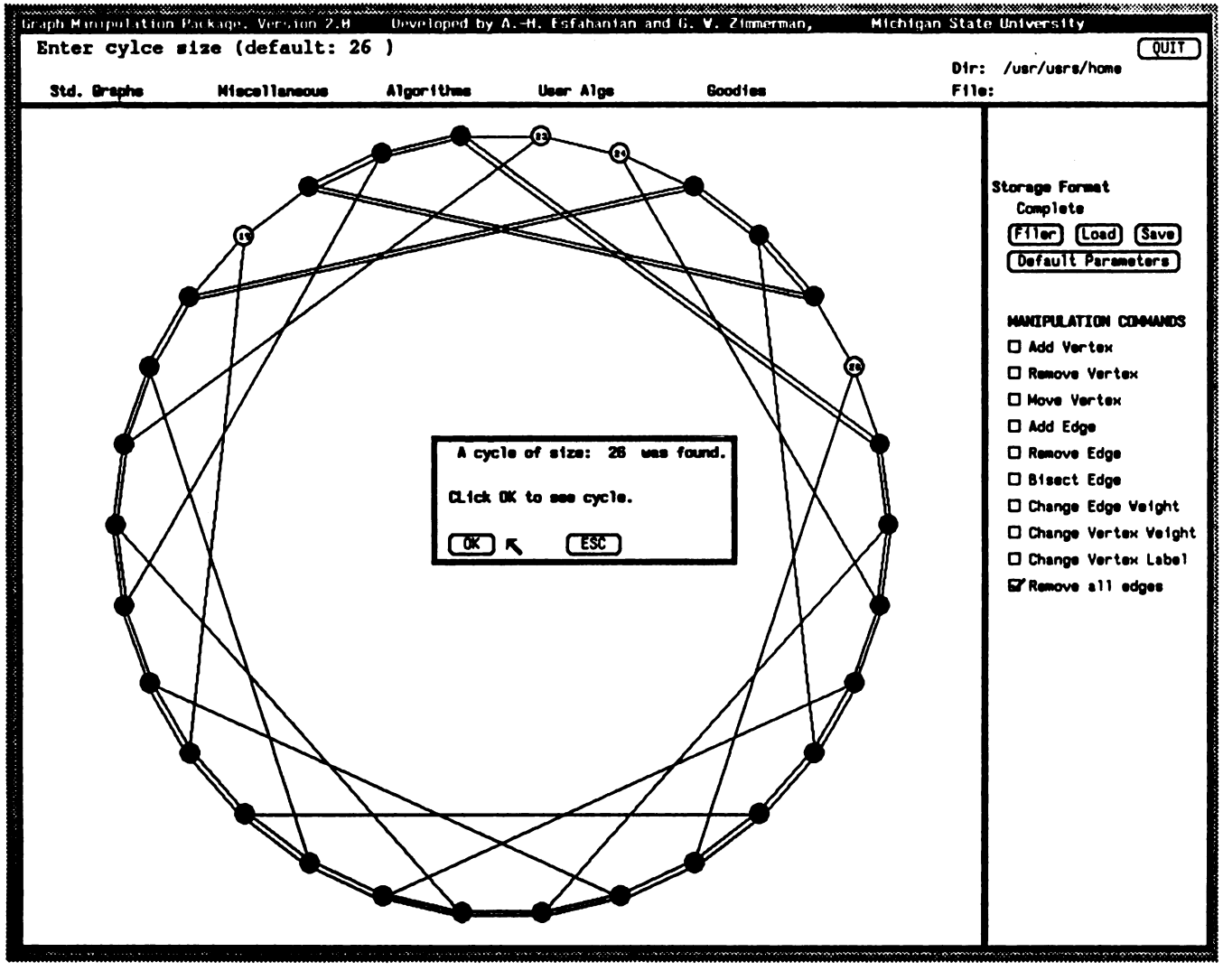

Figure 2-5. The Find Cycles algorithm is invoked on a Chordal Ring.

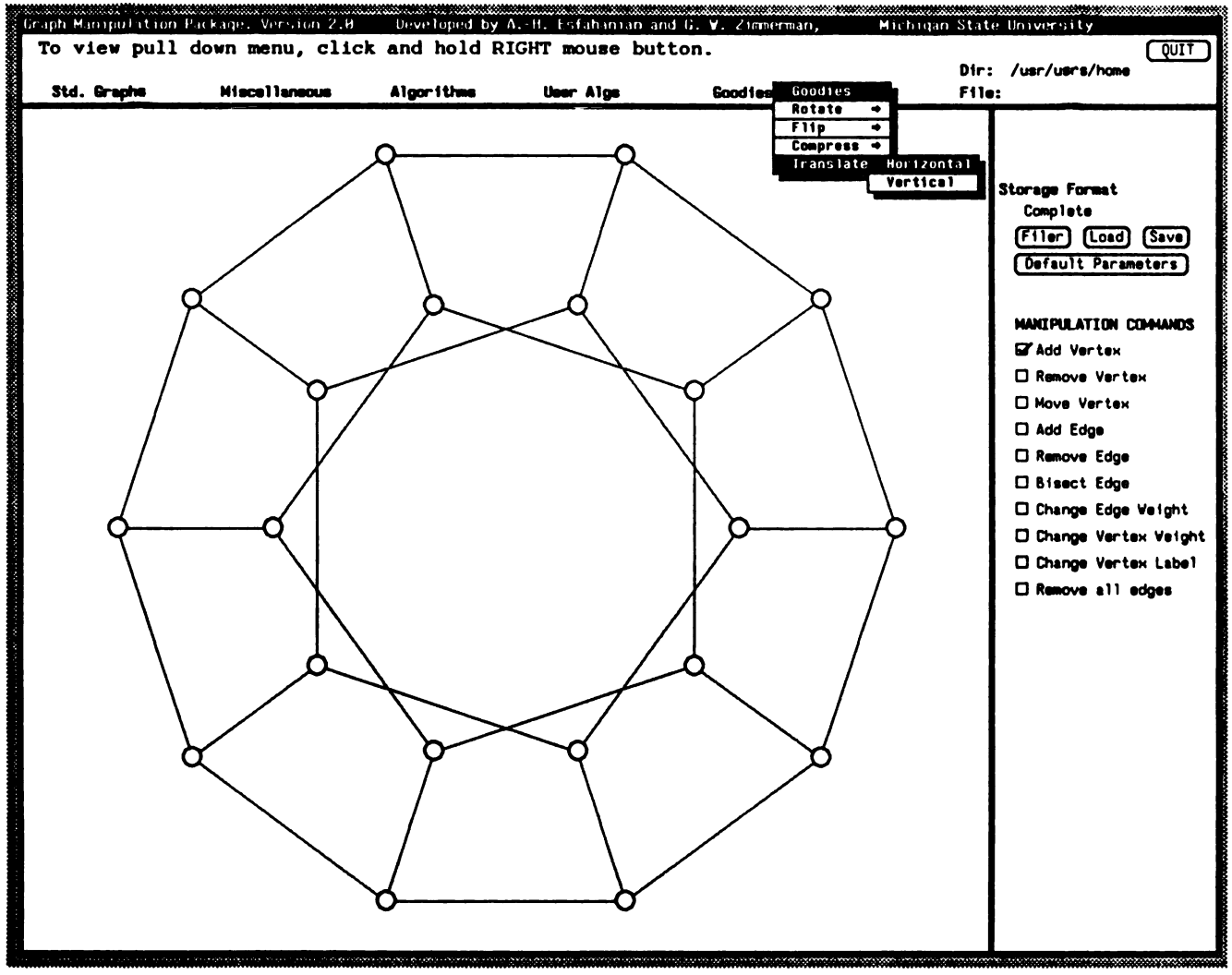

Figure 2-6. The Goodies Menu.

Chapter 3

Broadcasting in Multicomputers

One type of information exchange that often arises in MCs and communication networks in general (e.g. computer networks, spy networks) is referred to as *broadcasting*. Here, one member of the network, called the originator, wishes to send a message to all the other members in the network, as rapidly as possible. The underlying topology of the network plays a significant role in how broadcasting should be done. In some networks, by default, every message transmitted is received by all members of the network, whereas in some others, a transmitted message is received by no more than one other member of the network. These networks are respectively referred to as *broadcast networks* and *point-to-point* networks [Tane88]. Here we are concerned with the problem of broadcasting MultiComputer networks as defined in Chapter 1. However, since our focus is at the topological level, our discussion is also applicable to general point-to-point networks, and we hereafter refer simply to networks.

There are many applications of broadcasting in such networks. In general terms, broadcasting is necessary whenever some information must be relayed to all the other members of the network. Some specific examples include: synchronizing processors in

a distributed network, reconfiguring processors to achieve a desired logical interconnection, diagnosing the state of a distributed system, and communicating results from parallel algorithms computations [Prad85].

The basic idea in broadcasting is to keep informing uninformed members of the network until all member are informed, however, which uninformed members and how many of them can be informed at a time are usually restricted by the network technology and/or application environment. This has led to defining different types of broadcasting. In this chapter, we first survey the existing work on broadcasting in a network. We then discuss the difficulties that arise in implementing broadcasting and review the known algorithms for broadcasting. We will examine the role that trees play in broadcasting and we present a distributed broadcasting algorithm for a particular class of networks based on Binary DeBruijn Graphs.

3.1. Background

We first consider the broadcasting process in general. In this process, one node called the originator wishes to send a message to all the other nodes in the network. The originator begins the process by making a "call" to another node in the graph, informing it of the message. Subsequently, the informed nodes call their uninformed neighbors and the process continues until all nodes in the network are informed. It has been suggested [FHMP79] that each node be involved in at most one call during each time interval, although calls between distinct pairs of nodes may take place concurrently. We assume that all calls take the same amount of time; the time interval during which a call takes place will be referred to as a *time step* or simply *step*. Thus during the first step of the broadcasting process, the originator calls another node in the graph. At the end of the first step, two nodes are informed. Since there is no benefit gained from a node receiving the same message twice, we specify that each node (except the originator) be the receiver of exactly one call during the broadcast.

Several types of broadcasting have been distinguished in the literature [Farl80]. In *local broadcasting* a node may only call one of its neighbors. In *line broadcasting* a node may call any other node to which it is connected by a path; however, the edges used to make the call must not be used by any other call during that step. *Path broadcasting* is similar to line broadcasting, except that it is further required that no node is used in more than one call in any time step. The difference between line and path broadcasting is that in the former, a node may allow several calls to pass through during any step, whereas in the latter each node may participate in at most one call per step. In both cases, however, only the starting node in a broadcast path need be informed; all the intermediate nodes in the path can act as a switch and relay the message without being aware of its contents. This assumption may not be very realistic; if a node is to relay a message it might as well examine its contents since, in this context, the same message will be sent to it eventually. Here, we will consider only local broadcasting and unless stated otherwise, we will use the terms broadcasting and local broadcasting interchangeably.

The broadcast process is illustrated in Figure 3-1. The top graph represents the topology of a communication network. Broadcasts originating from nodes e and f are shown in the middle and bottom graphs respectively. The edges used in the broadcast are highlighted with arrows to indicate the sender/receiver and are labeled with the time step in which they are used. Unused edges are shown as dashed lines. Note that the broadcast was completed in five time steps in the middle graph; the lower graph used only four.

It is clearly desirable to complete broadcasting in a minimum number of time steps. For a general graph with N nodes, the minimum amount of time to complete a broadcast is at least $\lceil \log N \rceil$ (all logarithms are in base 2). This result follows from the fact that the number of informed nodes can at most double after each step of the process and may be proved formally by a simple inductive argument [FHMP79]. Clearly, the "location" of the originator within the graph affects the amount of time needed; an originator which is "in the middle" of the graph may require fewer time steps than one on the "periphery".

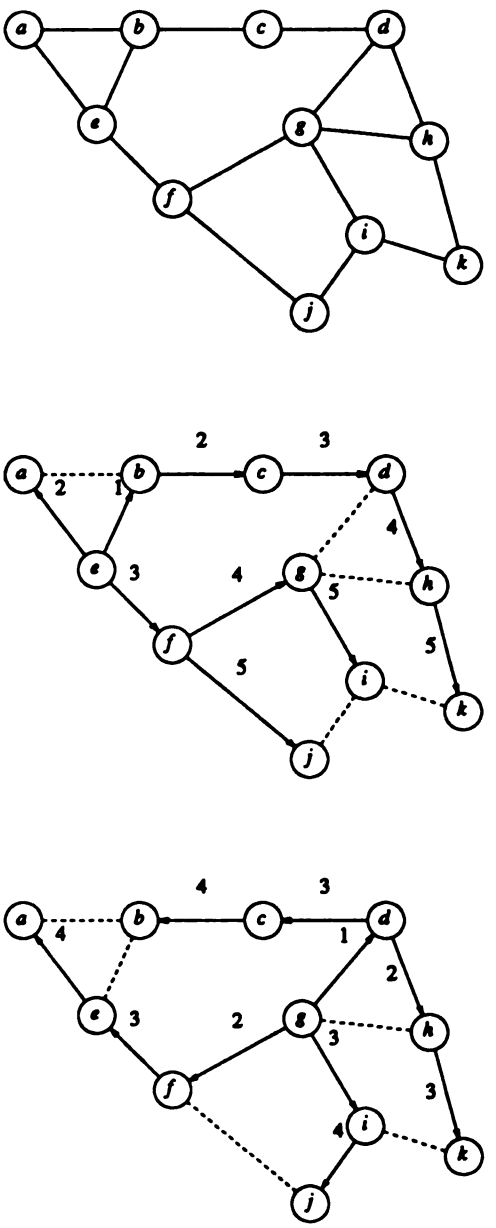

Figure 3-1. Two example broadcasts in a graph.

For example, in Figure 3-1, broadcasting from node a will require more time than broadcasting from node g. In addition, the sequence in which an informed node informs its uninformed neighbors affects the number of time steps for a particular broadcast. In the middle graph of Figure 3-1, e sends messages to nodes b, a, and f, in that order. However, it is not difficult to see that the number of steps can be reduced to four by using the order f, b, a. These considerations motivate the following definitions.

Definition. 3.1 The minimum time (in number of time steps) required to complete broadcasting in a graph G, when node v is the originator is denoted $bt(G:v)$. The minimum is taken over all possible orderings of calls.

Definition. 3.2 Let $bt(G)$ be the best possible broadcast time for a particular graph G. We have $bt(G) = \min\{bt(G:v) | v \in V\}$. Note that $bt(G)$ is not necessarily equal to $\lceil \log |V(G)| \rceil$.

Definition. 3.3 Let $BT(G)$ be the worst possible broadcast time for a particular graph G. We have $BT(G) = \max\{bt(G:v) | v \in V\}$.

Definition. 3.4 The broadcast center of a graph G, $BC(G) = \{v \in V \mid bt(G:v) = bt(G)\}$, is the set of all nodes which when chosen as the originator, can achieve the best possible broadcast time for G. This set is clearly nonempty and may have more than one element.

Definition. 3.5 A *calling schedule* for a broadcast is a set of statements of the form "node i calls node j during time step t". The calling schedule is a *legal calling* schedule if it satisfies the criteria [Farl79]:

i) During a given time step, each node may participate in at most one call.
ii) For each call scheduled for step t, the sender is either the originator or was called during step s, $1 < s < t$.

A graph $G(V,E)$ is said to be a *minimum broadcast* graph if $bt(G) = \lceil \log |V(G)| \rceil$. In addition, when $BC(G) = V(G)$ then G is called a *uniformly minimum broadcast* (UMB) graph. The above graphs are defined differently in [FHMP79], however we believe our definitions are more consistent with the rest of the related literature. By

definition, a broadcast informs all the nodes in the graph and to accomplish this, it uses some of the edges in the graph, each edge being used at most once. The subset of edges $E' \subseteq E$ used in the process of broadcasting in $G(V,E)$ *induces* [Hara72] a rooted spanning tree $T(V,E')$ of G. Further, the root of the tree is the originator of the broadcast and there is a direction associated with the edges in the tree. A tree $T(V,E)$ is a *minimum broadcast tree* (MBT) if it is a minimum broadcast graph.

Broadcasting has received attention in the literature as both an abstract graph theory problem and as a practical problem in communication networks. The problem of constructing UMB graphs is considered in [Farl79] and algorithms for their construction are also presented. Designing UMB graphs with the minimum number of edges is discussed in [Farl79, FHMP79] and solutions are given for a number of cases. In [FHMP79], such graphs are termed "minimum broadcast graphs" and a catalog of these graphs with 15 or fewer nodes is provided, along with proofs showing them to possess the minimum possible number of edges. This work has led us to consider what characterizes a minimum broadcast graph, *i.e.*, what are some of the necessary and/or sufficient conditions for a graph to be a minimum broadcast graph? For general graphs no work has been done in this regard.

In [Pros81], an algorithm is given to *recognize* minimum broadcast trees and to construct all rooted minimum broadcast trees for a given number of nodes. An algorithm for completing line broadcasting in minimum time is presented in [Farl80]. The algorithm also produces a legal calling schedule for the broadcast. In [FaHe79], local broadcasting in finite and infinite grid graphs is considered. The minimum times to complete broadcasting are derived for several special cases of grid graphs; among these is the graph which represents the topology of the Illiac IV-type array processor [Barn68]. A conjecture is made as to an upper bound on the number of nodes in an infinite grid that may become informed after a finite number of steps. A survey of much of the work on broadcasting and the more general problem known as "gossiping" is provided in [HeHL88].

3.2. Broadcasting in General Graphs

In a general connected graph $G(V,E)$, the problem of finding $bt(G:v)$ and a corresponding legal calling schedule for an arbitrary $v \in V$, has been shown to be NP-hard [SlCH81]. The same is true for determining $BC(G)$. In some circumstances, once a "good" broadcasting schedule has been obtained, there is no need to determine a new one. However, in other situations this may not be the case. For example, the topology of a point-to-point network may change, either physically due to failure/recovery of a computing element or communication channel, or logically due to reconfiguration. With either of these changes, a new legal broadcast schedule may need to be determined. Additionally, as an attempt to increase overall efficiency, it may be desirable to determine alternative broadcast schedules dynamically in response to traffic flow in the network induced by specific applications. The above concerns have motivated researchers to investigate heuristic algorithms for broadcasting in general graphs.

A class of heuristic algorithms for local broadcasting in general graphs is presented in [ScWu84]. The heuristic algorithms presented are all variations on one basic process: broadcasting in a graph G is done by successively generating maximum matchings in bipartite graphs derived from G while optimizing a weight function. Different choices of weight functions yield the variations. The performance analyses of these heuristics are yet to be seen.

Although the problem of determining $bt(G:v)$, $bt(G)$, $BT(G)$ and $BC(G)$ of a general graph G are NP-hard, this is not the case when G is a tree. In [SlCH81], algorithms have been developed for determining $BC(T)$ and for finding a broadcast schedule requiring $bt(T)$ time steps for a given tree T. We remind the reader that, by definition, the edges used in a broadcast in a given graph G induces a spanning tree of G. Therefore, to find $bt(G)$ we need essentially to select a spanning tree T of G such that $bt(T) = bt(G)$, and we know from Section 3.1 that such a spanning tree exists. In the absence of any selection criterion, we may need to examine all the spanning trees of the graph in order

to find a suitable one. This method, has the obvious drawback that the number of spanning trees of a graph may be an exponential function in the number of nodes, making the selection process very costly if not impractical. An alternative would be to examine a subset of spanning trees (which is the idea behind the existing heuristics) with the hope that it will contain a spanning tree T such that $bt(T) = bt(G)$.

The above discussions motivated us to examine the role that trees play in the broadcasting process. If we let $ST(G)$ be the set of all distinct spanning trees of a graph G and $BST(G) = \{T \in ST(G) \mid bt(T) = bt(G)\}$, we are interested in the comparative sizes of these sets.

3.3. Broadcasting in Trees

We have noted that trees play a fundamental role in the broadcasting process. In this section we will examine this role beginning with a discussion of minimum broadcast trees. In particular, we discuss a special class of MBTs - those whose order is equal to a power of two. The properties of this class of trees are used to establish several theorems, which characterize general MBTs. Finally, we examine the distribution of broadcast times for all trees of order $|V(T)| \leq 28$. We will then be able to state some general ideas for a heuristic algorithm for broadcasting.

3.3.1. Minimum broadcast trees

In this section, $T(V,E)$ will denote a tree with $|V| = N$. Recall that a tree T is an MBT if $bt(T) = \lceil \log N \rceil$ and $BC(T)$ is the set of all nodes u in T such that $bt(T:u) = bt(T)$. Each $u \in BC(T)$, along with the direction induced on the edges of T by its corresponding broadcast process, defines a rooted tree on T. We will refer to such trees as *rooted* minimum broadcast trees. Note that two rooted MBTs may have the same underlying tree. Figure 3-2 shows an example of three different rooted MBTs on nine nodes. In each case, the uppermost node is the originator of the broadcast. Each of these

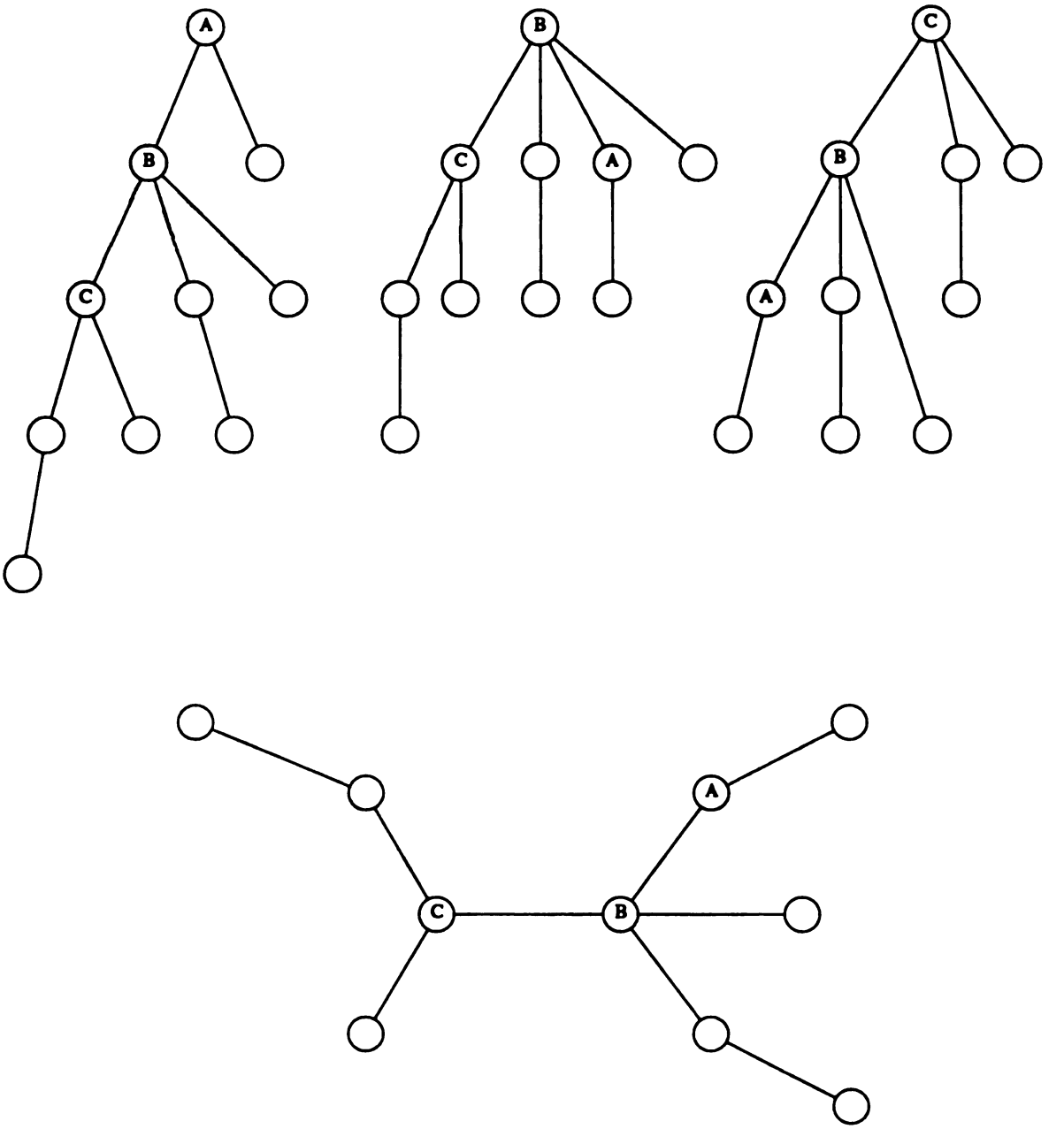

Figure 3-2. Three rooted MBTs and their common underlying free tree.

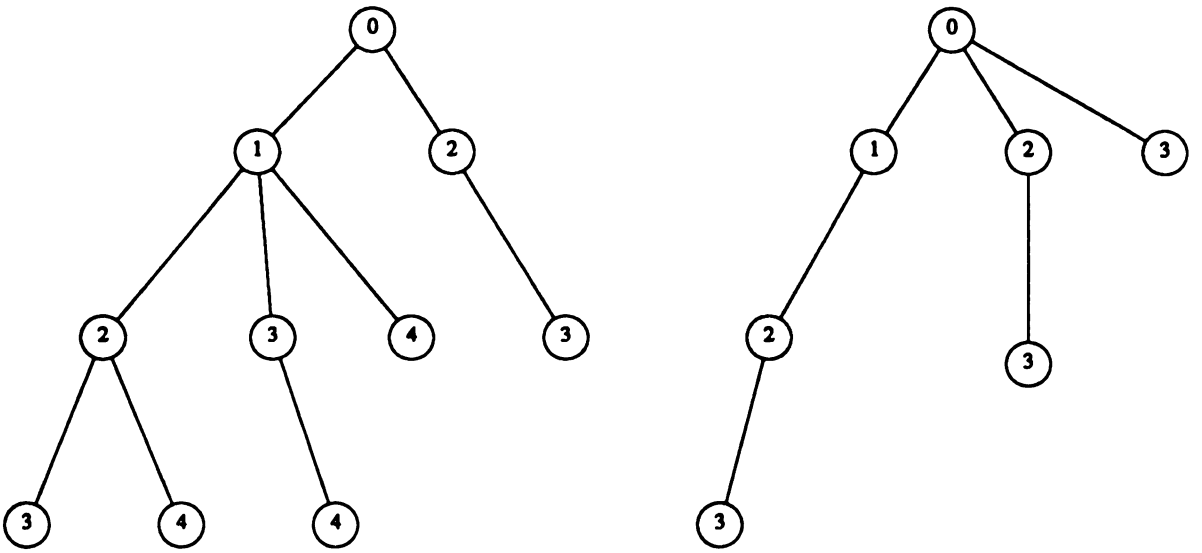

Figure 3-3. Two examples of MBTs.

rooted trees shares the same underlying free tree which is also shown in the figure. The labels (A,B,C) indicate how each rooted tree maps to the underlying tree. An algorithm has been developed for constructing and counting all rooted MBTs on N nodes [Pros81]. We are, however, interested in counting the number of distinct MBTs rather than rooted MBTs. We denote by $SMBT(N)$ the set of all distinct MBTs on N nodes. Figure 3-3 gives some examples of MBTs.

A special class of MBTs are those with $N = 2^k$ nodes ($k>0$). These trees contain the maximum number of nodes that can be informed in k time steps. The trees in this class (also known as binomial trees) are in a sense, completely defined with respect to many different properties, some of which are given below. We will utilize these properties in the characterization theorems for general MBTs in the next section.

Property 1. For each value of $k>0$, the set $SMBT(2^k)$ has exactly one element. That is, up to isomorphism, there is a unique minimum broadcast tree with 2^k nodes and we will denote it by $MBT(2^k)$. Figure e-4 illustrates some examples of trees from this class. The

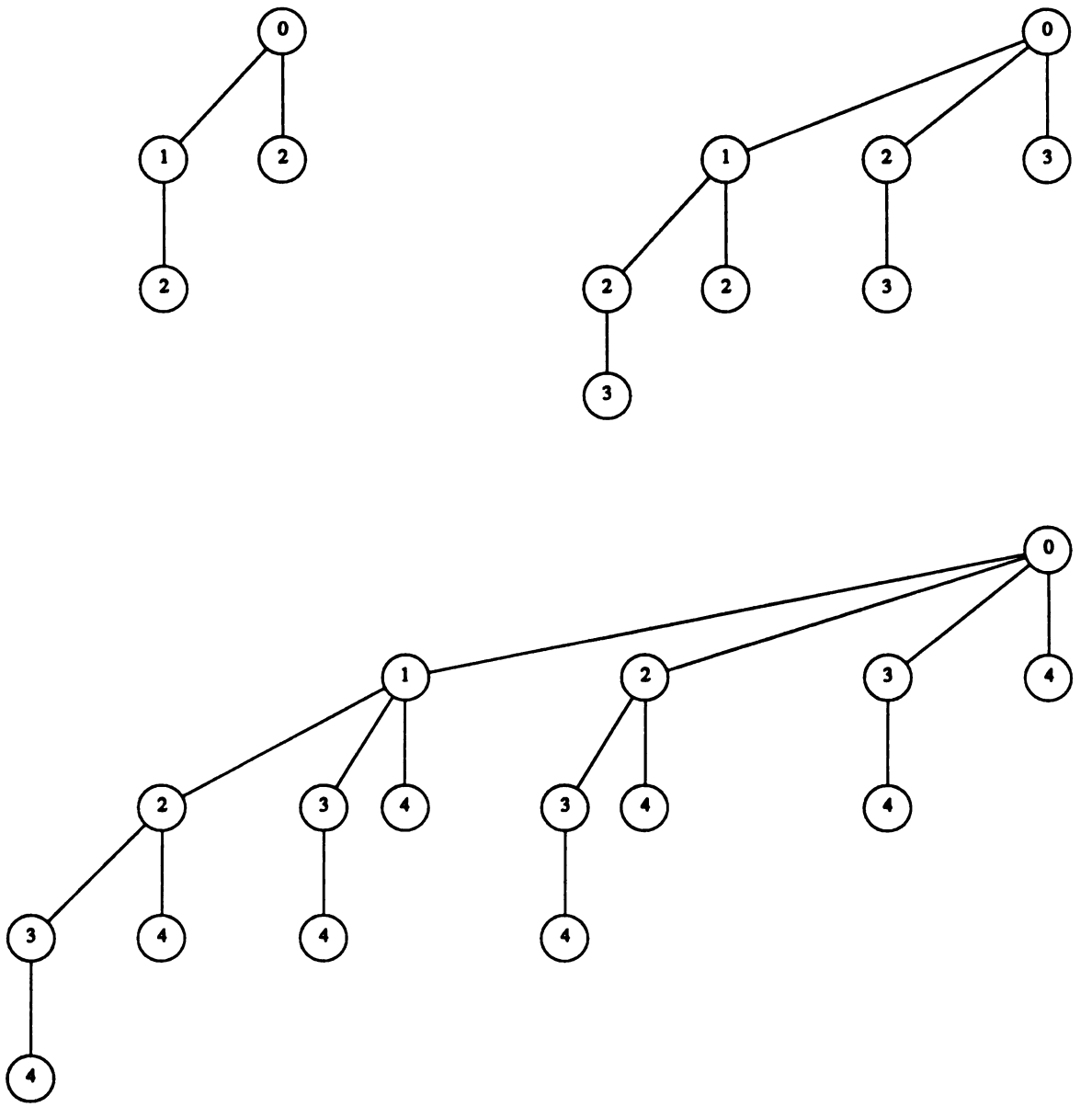

Figure 3-4. The unique trees $MBT(2^k)$, for $k = 2,3,4$.

labels indicate the time step in which each node becomes informed.

Property 2. $MBT(2^k)$ can be constructed by combining two copies of $MBT(2^{k-1})$; the tree can be decomposed into two symmetric halves. The construction is depicted in Figure 3-5. Let u be an originator in one copy of $MBT(2^{k-1})$ and v be an originator in a second copy of $MBT(2^{k-1})$. A single edge is added between the originators, yielding a tree with exactly 2^k nodes. To see that is also an MBT, choose u (or v arbitrarily) to be the originator for the new tree. Initially, u calls v, using 1 time unit. Since each of u and v is the originator in their respective subtrees, the remaining nodes may be informed in $k-1$ steps. It is clear that all trees $MBT(2^k)$ can be so constructed.

Property 3. The degree sequence of $MBT(2^k)$ is completely determined and can be computed recursively. In particular, there are two nodes of degree k; one of which must be the originator of the broadcast; the other must be the first node to become informed in the broadcast process. And, for each d, $1 \le d \le k-1$ there are 2^{k-d} nodes of degree d.

Property 4. There is exactly one calling schedule for $MBT(2^k)$, once the originator is fixed.

Property 5. The number of nodes at any level λ in $MBT(2^k)$is given by $\binom{k}{\lambda}$, hence the name binomial tree.

Property 6. The number of levels in $MBT(2^k)$ is exactly k.

Property 7. The number of leaves in $MBT(2^k)$ is exactly 2^{k-1}.

3.3.2. Characteristics of general minimum broadcast trees

We now present some results for general MBTs in the form of several theorems. These theorems represent some necessary (and sufficient) conditions for a tree to be a minimum broadcast tree. These results will serve as a first step in characterizing minimum broadcast graphs.

Theorem 3-1: A tree $T(V,E)$ of order N is a member of $SMBT(N)$ if and only if T is a subgraph of $MBT(2^k)$ where $2^{k-1} < N \le 2^k$.

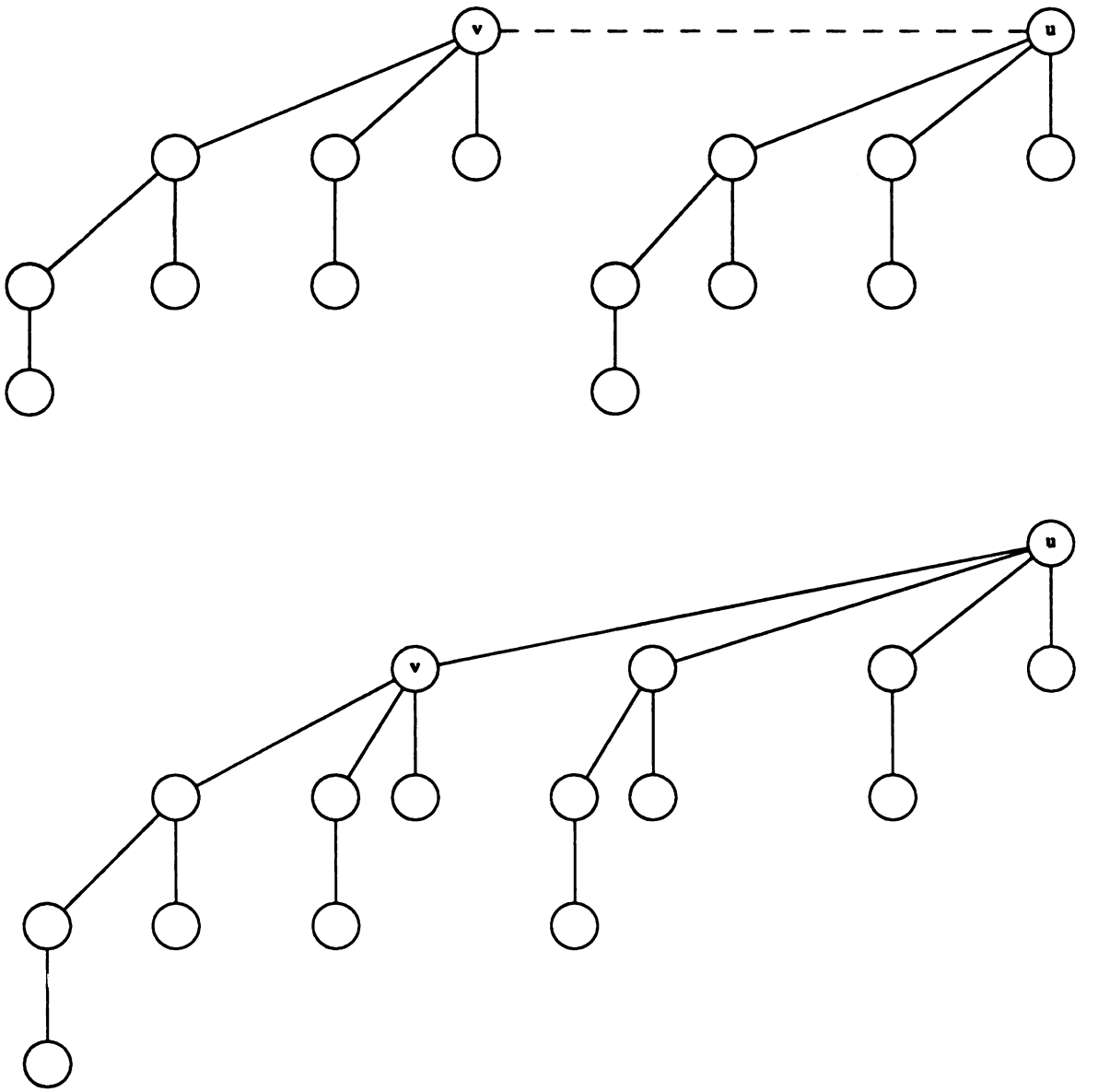

Figure 3-5. *MBT* (2^4) is constructed from two copies of *MBT* (2^3).

Proof: ($\rightarrow$) By definition, there is a legal calling schedule C, which will complete the broadcast in T in $k = \lceil \log N \rceil$ time steps. Begin the broadcasting. Let step τ be the first step during which a node has no neighbors left to inform and $S(\tau) = \{v \in V \mid v$ and all of its neighbors were informed before step $\tau\}$. For each node $v \in S(\tau)$ add a new node u_v to V and add a new edge to E, connecting u_v and v. Inform all the newly created nodes. Continue this process during each subsequent step through time step k. It is clear that in each step we inform as many nodes as possible (by adding new nodes where necessary, it is assured that the number of informed nodes must double) and that all nodes are informed in k steps. But $MBT(2^k)$ is the only tree with these properties. Thus we have constructed $MBT(2^k)$ by adding nodes and edges to T and hence the tree T is a subgraph of $MBT(2^k)$.

($\leftarrow$) Suppose $T \subseteq MBT(2^k)$. By property 4, there is a unique calling schedule C' for $MBT(2^k)$. Now remove all elements of C' which refer to a node not in T. We thus obtain a legal calling schedule C, $C \subset C'$ for T which completes the broadcast in k time steps. Thus $T \in SMBT(N)$. ■

Observe that the above proof implies that we can embed T in $MBT(2^k)$ in such a way that the originator of T is embedded onto the originator of $MBT(2^k)$. It should be noted that it is not necessarily the case that $MBT(2^{k-1})$ is a subgraph of an MBT on N vertices, where $2^{k-1} < N \leq 2^k$. This is illustrated in Figure 3-6 where an MBT with 9 nodes (right) is constructed by inserting a node (labeled C) into the tree $MBT(2^3)$ (left). It is clear from the figure that the original tree (left) is not a subgraph of the new one, illustrating that an MBT need not have $MBT(2^k)$, $k > 2$ as a subgraph.

Theorem 3-2: Let $T(V,E) \in SMBT(N)$. Then $\Delta(T) \leq \lceil \log N \rceil$.

Proof: Let $v \in V$ be a node whose degree $d(v) > \lceil \log N \rceil$. Suppose $v \in V$ is an originator of T. Since a tree is acyclic, v must inform each of its adjacent nodes and this takes more than $\lceil \log N \rceil$ steps, contradicting the definition of T. Now suppose v is not an originator. It takes at least 1 time step to inform v, using one of the edges incident to v, v

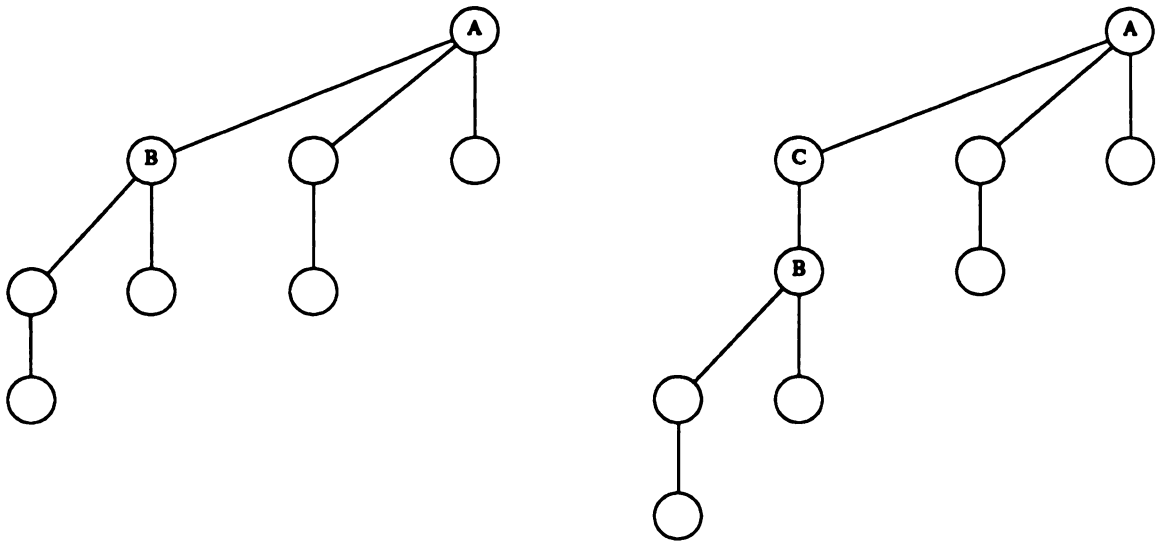

Figure 3-6. An MBT not having $MBT(2^3)$ as a subgraph.

then has $d(v)-1$ uninformed neighbors. Since the only path to these nodes is through v, v must send the message to each of them in turn. This requires $d(v)-1$ calls. Thus the time to complete the broadcast is at least $d(v) > \lceil \log N \rceil$, again violating minimality. Thus no such v exists. ■

Note that the converse of the above theorem is not necessarily true; it is easy to construct trees with small maximum degree, which are not MBTs. For example, consider a path P of order 2^k, $k>2$. To be an MBT, it would have to be possible to complete a broadcast in P in $t \leq k$ time steps. It is clear that this cannot be done; the reason being that each vertex (other than the originator) may only inform one other vertex, and there are more vertices in the tree than can be informed in the required number of time steps.

An interesting related question is: how many nodes can be informed in a tree with maximum degree Δ after t time steps? In lemma 3.1, a recurrence relation is established which relates the maximum number of nodes in a tree T which can be informed *during*

time step τ, to the maximum degree of the tree, Δ.

Lemma 3-1 : Let T be a tree with maximum degree Δ and $MAX(\tau,\Delta)$ be the maximum number of nodes in T which can be informed during time step τ of a local broadcast in T. Then the following recurrence relation holds.

$$MAX(\tau,\Delta) = \begin{cases} 2^{\tau-1} & 1 \le \tau \le \Delta \\ MAX(\tau-1,\Delta) + MAX(\tau-2,\Delta) + \cdots + MAX(\tau-\Delta+1,\Delta) & \tau > \Delta \end{cases}$$

Proof. The correctness of the recurrence relation may be seen as follows. Consider an infinite rooted tree in which all nodes have degree Δ. With the root of the tree being the originator, we wish to know how many nodes of this tree may be informed during time step τ. During each step $\tau \le \Delta$, each informed node has unused edges which may be used to inform an uninformed node. Thus we generate $MBT(2^{\tau})$ and consequently $2^{\tau-1}$ nodes are informed during time step τ. Now, consider any $\tau > \Delta$. Every node which is first informed at time step $\tau-\Delta+1$ has exactly $\Delta-1$ edges by which it can inform other nodes. Since it may inform only one node per time step, it will inform one node during time steps $\tau-\Delta+2, \tau-\Delta+3, \ldots, \tau$. Thus every node which becomes informed at step $\tau-\Delta+1$ will inform a node during time step τ. Similarly, every node which becomes informed at time step $\tau-\Delta+k$ will inform a node during steps: $\tau-\Delta+k+1, \tau-\Delta+k+2, \ldots, \tau$. This yields the stated recurrence relation. ■

The relation of lemma 3-1 can be used to determine the maximum number of nodes which can be informed in a tree T with maximum degree Δ, in t time steps. Let $M(t,\Delta)$ represent this number. Then we have

$$M(t,\Delta) = \sum_{i=1}^{t} MAX(i,\Delta) \qquad 3.1$$

Table 3-1 shows the values of $M(t,\Delta)$ for $\Delta = 2,3,4,5$ and $t = 1, \ldots, 20$. The entries in the second through fifth columns of the table are the values of $M(t,\Delta)$. The last column gives the lower bound on the number of nodes which must be informed in order for a tree

Table 3-1. The values of $M(t,\Delta)$.

t	$\Delta = 2$	$\Delta = 3$	$\Delta = 4$	$\Delta = 5$	Minimum
0	1	1	1	1	0
1	2	2	2	2	1
2	4	4	4	4	2
3	6	8	8	8	4
4	8	14	16	16	8
5	10	24	30	32	16
6	12	40	56	62	32
7	14	66	104	120	64
8	16	108	192	232	128
9	18	176	354	448	256
10	20	286	652	864	512
11	22	464	1200	1666	1024
12	24	752	2208	3212	2048
13	26	1218	4062	6192	4096
14	28	1972	7472	11936	8192
15	30	3192	13744	23008	16384
16	32	5166	25280	44350	32768
17	34	8360	46498	85488	65536
18	36	13528	85524	164784	131072
19	38	21890	157304	317632	262144
20	40	35420	289328	612256	524288

to be an MBT. For example, with $t=5$, 17 or more nodes must be informed for the tree to be an MBT. If 16 or fewer are informed, then by definition, this cannot represent an MBT. The table indicates in particular, that the largest MBT with maximum degree 2 has 6 nodes. This is because with $\Delta=2$ and $t=4$, at most 8 nodes can be informed and thus, by definition, the tree will not be an MBT. The table also shows that if a tree with maximum degree 3 has more than 66 nodes, it is not possible to complete broadcasting in 7 time steps. Thus no MBT with maximum degree 3 may have more than 66 nodes. Similar claims may be made regarding trees with higher maximum degrees.

Theorem 3-3: Let $T(V,E) \in SMBT(N)$ and λ be the number of levels in T. Then

$$\left\lceil \frac{\lceil \log N \rceil}{2} \right\rceil \leq \lambda \leq \lceil \log N \rceil.$$

Proof: The right hand inequality can be seen by recalling that by Theorem 3-1, T is a subgraph of $MBT(2^k)$, where $k = \lceil \log N \rceil$. By property 6, the number of levels in $MBT(2^k)$ is $k = \lceil \log N \rceil$. We prove the left hand inequality by contradiction. Suppose $\lambda < \lceil k/2 \rceil$. First, note that T cannot have fewer than $2^{k-1}+1$ nodes (otherwise it will not belong to $SMBT(N)$). By property 5, the number of nodes in $MBT(2^k)$ has a binomial distribution over the levels of the tree. Embed T onto $MBT(2^k)$ and then remove all nodes in levels $\lceil k/2 \rceil$ and greater. Now we consider two cases.

Case 1: k is odd. The distribution of nodes is *symmetric* with respect to the levels in the tree. Here we have removed exactly half of the nodes from $MBT(2^k)$, so that the number of remaining nodes is 2^{k-1}.

Case 2: k is even. The distribution of nodes is *asymmetric*. Here we have removed the middle level as well as the levels below it, and the number of nodes remaining is less than 2^{k-1}.

In both cases, we must have removed nodes from T in the pruning process, since the pruned tree has fewer nodes than T is required to have. This yields the contradiction. ■

Theorem 3-4: Let $T(V,E) \in SMBT(N)$ and ρ be the number of leaves in T. Then, $\rho \leq 2^{\lceil \log N \rceil - 1}$.

Proof: Let $k = \lceil \log N \rceil$ and embed T within $MBT(2^k)$. Recall property 7. Now prune $MBT(2^k)$ by removing leaves, one at at time, to yield T. Clearly, the pruning process cannot increase the number of leaves. ■

3.3.3. Broadcast times for general trees

We have seen that for an arbitrary tree T, $\lceil \log N \rceil \leq bt(T) \leq N-1$, and by applying some of the above results we can further refine these bounds. Even so, this represents a large range of broadcast times. We examined all possible trees on N nodes ($1 \leq N \leq 10$) and have noticed that in a large percentage of these, broadcasting can be done in the optimal or near optimal number of steps. This motivated us to investigate the distribution

of broadcast times for all the trees of a given order.

In [WROM86], an algorithm is presented which generates all free trees of a given order. Combining this algorithm with the algorithm to compute $bt(T)$ from [SlCH81], we wrote a computer program to calculate the distribution mentioned above, for all orders N, $4 \leq N \leq 28$. Verifying the correctness of the program was done by carefully determining that each algorithm was accurately translated into the C programming language and by comparing the results of the program for trees up to order 10 with previously known results. The results are given in Table 3-2, from which we make the following observations.

Table 3-2. Distribution of broadcast times for all trees, T, $4 \leq |T| \leq 28$.

	Order of Tree															
τ	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	2	4	2	1	0	0	0	0	0	0	0	0	0	0	0
4	0	1	1	7	17	28	42	46	45	29	16	4	1	0	0	0
5	0	0	1	1	3	14	52	147	370	788	1543	2727	4516	6867	9758	12715
6	0	0	0	1	1	3	7	30	105	390	1293	3935	10970	28407	69110	159211
7	0	0	0	0	1	1	3	7	19	63	229	848	3134	10951	36354	114449
8	0	0	0	0	0	1	1	3	7	19	47	149	494	1840	6974	26142
9	0	0	0	0	0	0	1	1	3	7	19	47	127	359	1136	3977
10	0	0	0	0	0	0	0	1	1	3	7	19	47	127	330	926
11	0	0	0	0	0	0	0	0	1	1	3	7	19	47	127	330
12	0	0	0	0	0	0	0	0	0	1	1	3	7	19	47	127
13	0	0	0	0	0	0	0	0	0	0	1	1	3	7	19	47
14	0	0	0	0	0	0	0	0	0	0	0	1	1	3	7	19
15	0	0	0	0	0	0	0	0	0	0	0	0	1	1	3	7
16	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	3
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3-2 (cont'd).

	Order of Tree								
τ	20	21	22	23	24	25	26	27	28
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	15334	16814	16818	15022	11966	8249	4900	2390	954
6	350268	739433	1505525	2963910	5657301	10480329	18865609	33004506	56130583
7	343884	989511	2741135	7336006	19039628	48059182	118308484	284672624	670891043
8	94433	328241	1097455	3542358	11075203	33654560	99685018	288543207	817941510
9	14990	57687	219887	816259	2936970	10237157	34637275	114047524	366427678
10	2732	8971	32196	122918	478692	1853958	7035486	26016218	93624639
11	889	2424	6938	21293	71723	262261	1011898	3985601	15655171
12	330	889	2378	6506	18162	53365	168262	578628	2146253
13	127	330	889	2378	6450	17577	48756	139014	417062
14	47	127	330	889	2378	6450	17510	47986	132470
15	19	47	127	330	889	2378	6450	17510	47907
16	7	19	47	127	330	889	2378	6450	17510
17	3	7	19	47	127	330	889	2378	6450
18	1	3	7	19	47	127	330	889	2378
19	1	1	3	7	19	47	127	330	889
20	0	1	1	3	7	19	47	127	330
21	0	0	1	1	3	7	19	47	127
22	0	0	0	1	1	3	7	19	47
23	0	0	0	0	1	1	3	7	19
24	0	0	0	0	0	1	1	3	7
25	0	0	0	0	0	0	1	1	3
26	0	0	0	0	0	0	0	1	1
27	0	0	0	0	0	0	0	0	1

Observations from Table 3-2.

1 The distribution is heavily skewed towards the lower values of τ. That is, in the vast majority of trees of a given order , broadcasting can be accomplished in near the theoretical minimum time.

2 A pattern is evident in the "tails" of each column. Most obvious is the diagonal of 1's at the end of each column. Moreover, moving from left to right, the entries in each tail become fixed after a certain point. For each column after the "10" column, the last 4 entries are always 7,3,1,1. In fact the number of "fixed entries" increases by one for each even tree order.

3.4. Broadcasting in Binary DeBruijn Graphs

Binary De Bruijn graphs (*BDG*) are graphs whose interconnections are determined by a simple coding scheme based on a numbering of the vertices in the graph. Networks modeled after binary De Bruijn graphs will be referred to as binary De Bruijn networks (BDNs). It has been shown that many network topologies can be embedded within BDNs. The *ring*, *one dimensional array*, *complete binary tree* and *shuffle exchange* networks are some examples of such networks. Each of these networks is particularly well suited for solving certain classes of problems [SaPr89]. For example, the shuffle exchange network admits efficient computation of FFT; a one dimensional array network is effective in solving problems in the pipeline class, among which is matrix-vector multiplication. A single BDN can be configured to act as any of a number of special purpose networks and thus can serve as a general purpose network for the efficient solution of a diverse set of problems. These facts make BDNs an attractive choice for the architecture of an MC.

We have noted some of the issues related to broadcasting in general graphs. For an arbitrary graph, determining an optimal broadcast schedule is NP-hard. We have given a lower bound for $bt(G)$ and noted that the value of $bt(G)$ is related to the maximum

degree of G. In this section, we consider the problem of broadcasting in BDNs. We first give a precise definition of binary De Bruijn graphs and some of their properties. Then we present a distributed algorithm for doing broadcasting in BDNs which requires $(2 \cdot \log_2 N)-1$ time steps where N is the number of nodes in the network.

3.4.1. Binary De Bruijn graphs

The *binary De Bruijn* graph, $BDG(n)$, has 2^n vertices and can be constructed using procedure 3.1. It is not difficult to see that each vertex $(b_{n-1},b_{n-2},\ldots,b_0)$ in $BDG(n)$ is adjacent to vertices $(b_{n-2},b_{n-3},\ldots,b_0,\alpha)$ and $(\alpha,b_{n-1},b_{n-2},\ldots,b_1)$ where $\alpha \in \{0,1\}$. This makes $BDG(n)$ a regular graph with the degree of each vertex equal to four.

Procedure 3.1 Construct $BDG(n)$.

1. Label the 2^n vertices using distinct binary n-tuples $(b_{n-1},b_{n-2},\ldots,b_0)$.
2. Connect (via an edge) each vertex $(b_{n-1},b_{n-2},\ldots,b_0)$ to vertices $(b_{n-2},b_{n-3},\ldots,b_0,0)$ and $(b_{n-2},b_{n-3},\ldots,b_0,1)$.

The fact that the $\Delta(BDG(n)) = 4$ (the number of connections at each processing element is fixed) and the diameter of $BDG(n)$ is equal to n means that the diameter of the network increases only logarithmically with the number of vertices. This makes $BDG(n)$ particularly attractive as an interconnection network. Additionally, many useful topologies are contained in $BDG(n)$ including a 2^n vertex linear array, a 2^n vertex ring, a (2^n-1) vertex complete binary tree, a $(3 \cdot 2^{n-2}-2)$ vertex tree machine, and a 2^n vertex one-step shuffle-exchange graph [SaPr89]. As mentioned in Section 3.4, each of these topologies is useful for solving various classes of problems. The fact that De Bruijn graphs can accommodate these topologies makes them an especially powerful topology.

The construction given in procedure 3.1 implies that the two vertices with n-tuple labels $(0,0,\ldots,0)$ and $(1,1,\ldots,1)$ have a loop edge. Further, the two vertices with labels $(0,1,0,1,\ldots,)$ and $(1,0,1,0,\ldots,)$, (i.e., alternating 0's and 1's) have 2 edges between them. In a practical situation, the loop edges would not be used and the doubled edge would be implemented as a single edge. Figure 3-7 shows $BDG(4)$ with the above modifications. In what follows, $BDG(n)$ will refer to this modified graph.

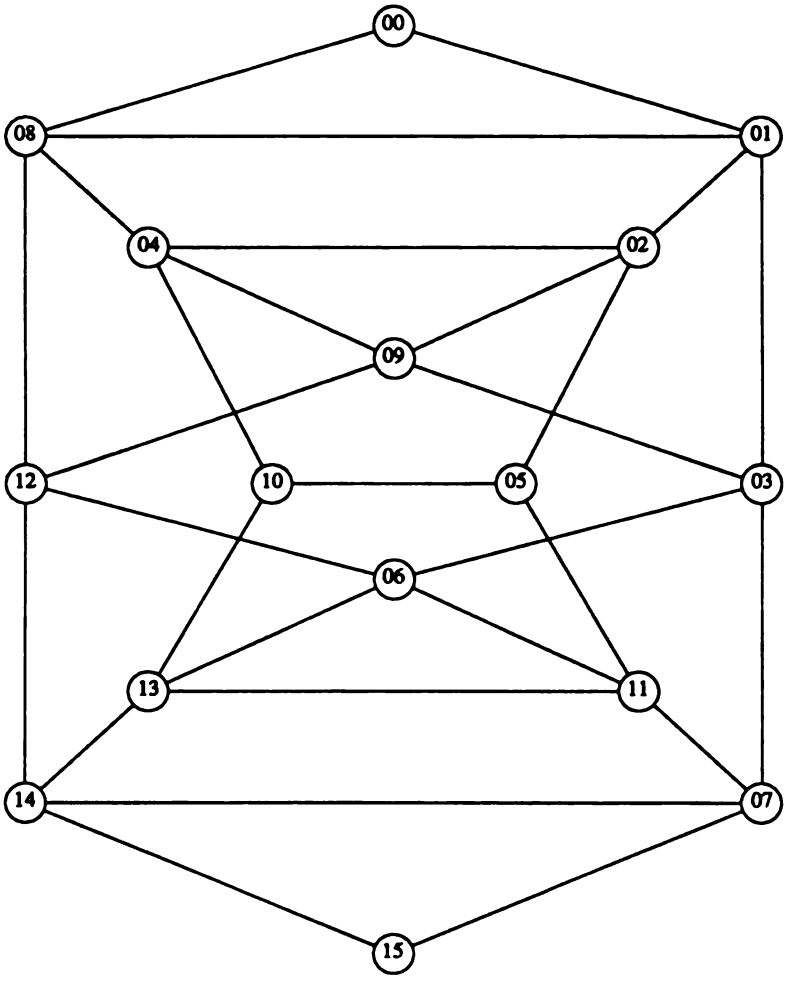

Figure 3-7. The graph $BDG(4)$.

3.4.2. A distributed broadcast algorithm for BDG(n)

In this section we present an algorithm which generates a calling schedule for broadcasting in $BDG(n)$. Observe that $bt(BDG(n)) \geq n$; however, in general $bt(BDG(n)) > n$. This is due to the fact that $\Delta(BDG(n))$ is a constant. The formula 3.1 from Section 3.3.2 can be used to rigorously prove that $BDG(n)$ is not a minimum broadcast graph in general. Our broadcasting algorithm for $BDG(n)$ generates a calling schedule that requires exactly $(2n-1)$ time steps to complete broadcasting, irrespective

of the choice of the originator. Also, the control of the broadcast process described by the algorithm is **distributed**, in the sense that each vertex can determine to whom it should transmit the message. It is only required that the label of the originator of the broadcast accompany the message.

Central to our broadcast algorithm is an integer valued function $BTS(A)$, called *broadcast time step function*, defined on binary m-tuples A. Let $A = (a_{m-1},a_{m-2},\ldots,a_0)$ and define $K(a_i,a_j)$, $0 \le i, j \le m-1$, as follows:

$$K(a_i,a_j) = \begin{cases} 2 & \text{if } a_i \oplus a_j = 0 \\ 1 & \textit{otherwise} \end{cases}$$

where $\oplus$ is the *binary Exclusive-OR* operation. BTS is used to determine the time step when a vertex becomes informed. Now, if $m \le 1$ then $BTS(A) = 0$, otherwise $BTS(A)$ is defined as:

$$BTS(A) = \sum_{i=0}^{m-2} K(a_i,a_{i+1})$$

Example 3.1 Let $A = (1,0,1,0,0,1,0)$. Then $BTS(A) = 1+1+2+1+1+1 = 7$. Note that the value of $BTS(A)$ is maximum if and only if $a_{m-1} = a_{m-2} = \cdots = a_0$, and the maximum value is $2(m-1)$.

Intuitively, our broadcasting algorithm proceeds as follows. Each vertex will inform at most two of its adjacent vertices. In particular, for a vertex $B = (b_{n-1},b_{n-2},\ldots,b_0)$, let $LSNE(B)$ and $LSCE(B)$ be two of its adjacent vertices whose labels, respectively, are

$$LSNE(B) = (b_{n-2},b_{n-3},\ldots,b_0,b_0) \quad \text{and} \quad LSCE(B) = (b_{n-2},b_{n-3},\ldots,b_0,\bar{b}_0)$$

where $\bar{b}_i$ is the bit complement of b_i. (*LSNE* and *LSCE* stand for Left Shift with Normal Extension and Left Shift with Complement Extension, respectively.) Vertex B will attempt to inform vertices $LSCE(B)$ and $LSNE(B)$, and in that order provided that these vertices have not already been informed. Observe that this scheme implies that a vertex

---

Algorithm 1.

1: Find the largest i such that

$$B = (b_{n-1},b_{n-2},\ldots,b_0) = (s_{i-1},s_{i-2},\ldots,s_0,b_{n-i-1},b_{n-i-2},\ldots,b_0)$$

2: /* check LSCE neighbor */
IF $LSCE(B) = S$ go to 4.
ELSE find the largest j such that

$$LSCE(B) = (s_{j-1},s_{j-2},\ldots,s_0,b_{n-j-2},b_{n-j-3},\ldots,b_0,\bar{b}_0)$$

3: IF $BTS(s_0,b_{n-j-2},b_{n-j-3},\ldots,b_0,\bar{b}_0) > BTS(s_0,b_{n-i-1},b_{n-i-2},\ldots,b_0)$
THEN inform $LSCE(B)$.

4: /* check LSNE neighbor */
IF $LSNE(B) = S$ STOP
ELSE find the largest k such that

$$LSNE(B) = (s_{k-1},s_{k-2},\ldots,s_0,b_{n-k-2},b_{n-k-3},\ldots,b_0,b_0)$$

5: IF $BTS(s_0,b_{n-k-2},b_{n-k-3},\ldots,b_0,b_0) > BTS(s_0,b_{n-i-1},b_{n-i-2},\ldots,b_0)$
THEN inform $LSNE(B)$. STOP.

---

can potentially get informed from two other vertices, since for any vertex $A = (a_{n-1},a_{n-2},\ldots,a_0)$ there are exactly two other distinct vertices X and Y such that either $LSNE(X) = LSNE(Y) = A$ or $LSCE(X) = LSCE(Y) = A$, namely,

$$X = (0,a_{n-1},a_{n-2},\ldots,a_1) \quad \text{and} \quad Y = (1,a_{n-1},a_{n-2},\ldots,a_1).$$

The function BTS will be used to check whether a vertex is already informed or not. We proceed by stating our broadcast algorithm, Algorithm 1, formally. An example of its execution is given, and then we prove a theorem which implies the correctness of the algorithm. It is assumed that the broadcast originator is vertex $S = (s_{n-1},s_{n-2},\ldots,s_0)$ and this information accompanies the broadcast message. Each vertex B, in $BDG(n)$, once informed of the broadcast message, will perform Algorithm 1.

Example 3.2 Let us trace its execution for vertex $B = (1,0,0,1)$ when the network is $BDG(4)$ and the originator of the broadcast is vertex $S = (0,0,1,0)$. In this case,

$LSNE(B)$ is the vertex (0,0,1,1) and $LSCE(B)$ is the vertex (0,0,1,0). The value of i found in step 1 is 2. Since $LSCE(1,0,0,1) = (0,0,1,0) = S$, execution will continue from step 4. In step 4, the value of k is found to be 1. Thus, in step 5 we have:

$$BTS(s_0,b_1,b_0,b_0) = BTS(0,0,1,1) = 2+1+2$$

and

$$BTS(s_0,b_1,b_0) = BTS(0,0,1) = 1+2,$$

and therefore since $BTS(s_0,b_1,b_0,b_0) > BTS(s_0,b_1,b_0)$, $LSNE(B)$ will be informed.

The broadcast process in $BDG(4)$ is illustrated in Figure 3-8. The arrows indicate the direction of the broadcast and highlight the spanning tree that is induced by the broadcasting. Note that, as in example 3.2, node 9 informs node 3 and not node 2, since node 2 is the originator.

Theorem 3-5. Let vertex $S = (s_{n-1},s_{n-2},\ldots,s_0)$ be the originator of a broadcast in $BDG(n)$. Also, let $A = (a_{n-1},a_{n-2},\ldots,a_0)$ be an arbitrary vertex in $BDG(n)$ and find the largest i such that

$$A = (a_{n-1},a_{n-2},\ldots,a_0) = (s_{i-1},s_{i-2},\ldots,s_0,a_{n-i-1},a_{n-i-2},\ldots,a_0).$$

Then, using Algorithm 1, vertex A is informed at the end of time step

$$T(A) = BTS(s_0,a_{n-i-1},a_{n-i-2},\ldots,a_0)$$

Moreover, at the end of time step $(2n-1)$ all vertices are informed.

Proof: We prove the theorem by induction on the number of time steps. Clearly, only $T(S) = BTS(s_0) = 0$. Now suppose the theorem is true for all vertices B with $T(B) \le k$, and let A be a vertex such that $T(A) = k+1$. Consider the vertex $X = (s_i,s_{i-1},\ldots,s_0,a_{n-i-1},a_{n-i-2},\ldots,a_1)$ where i is defined as in the statement of the theorem. Observe that vertex X is adjacent to vertex A in $BDG(n)$, since we either have $LSNE(X) = A$ or $LSCE(X) = A$. Also,

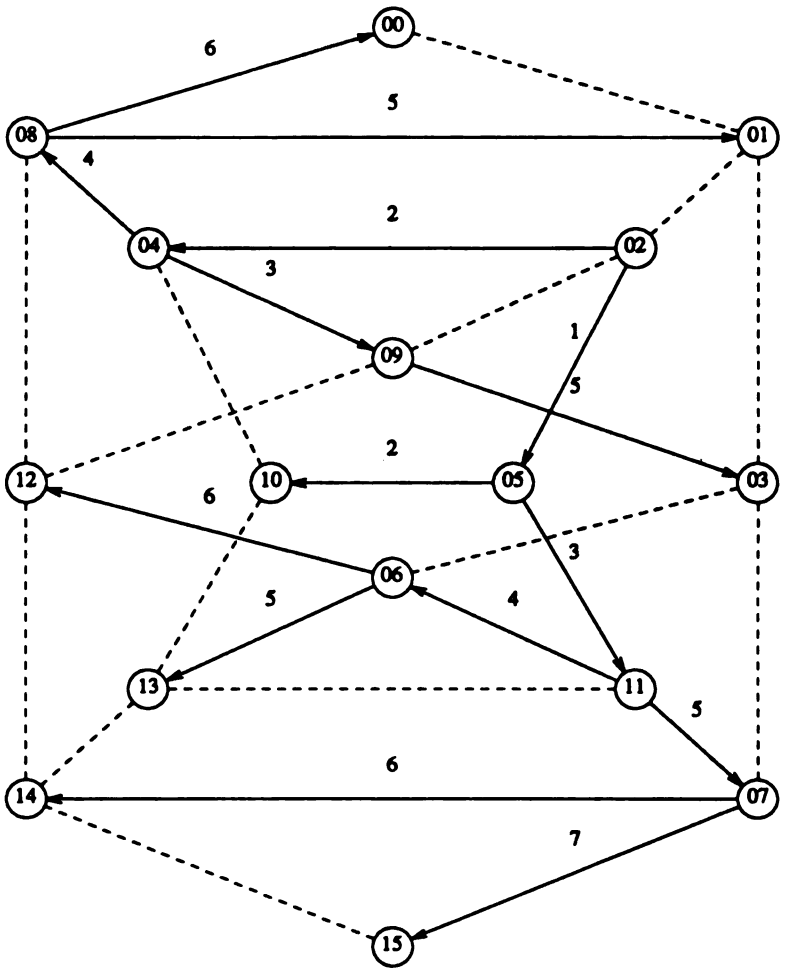

Figure 3-8. An example broadcast in $BDG(4)$ using Algorithm 1

$$\begin{aligned} T(A) &= BTS(s_0, a_{n-i-1}, a_{n-i-2}, \ldots, a_0) \\ &= BTS(s_0, a_{n-i-1}, a_{n-i-2}, \ldots, a_1) + BTS(a_1, a_0) \\ &= T(X) + BTS(a_1, a_0). \end{aligned}$$

Since $BTS(a_1, a_0) \geq 1$, we have $T(X) \leq k$. By the induction hypothesis, vertex X was informed by the end of time step $T(X)$. We now consider the following two cases:

Case 1: $a_1 \oplus a_0 = 1$. This means that $BTS(a_1, a_0) = 1$ and thus we have $T(X) = k$. Also, in this case, $LSCE(X) = A$, and therefore, by step 3 of Algorithm 1, vertex A will

be informed at the end of time step k+1.

Case 2: $a_1 \oplus a_0 = 0$. This implies $BTS(a_1,a_0) = 2$ and thus we have $T(X) = k-1$. In this case, $LSNE(X) = A$, and therefore, by step 5 of Algorithm 1, vertex A will be informed at the end of time step $k+1$.

In both cases, vertex A is informed at the end of time step $T(A)$. Note that when $A = (\bar{s}_0,\bar{s}_0,\ldots,\bar{s}_0)$, $T(A)$ is maximum. In this case, $T(A) = 2n-1$. Therefore, at the end of time step $(2n-1)$ all vertices are informed. ■

The above theorem and the inequalities in step 3 and step 5 of Algorithm 1 ensure that each vertex receives the broadcast message exactly once. Thus Algorithm 1 satisfies the criteria of local broadcasting.

Chapter 4

Fault-Tolerant Loop Architectures

The large number of components (processors, memory units, etc.) in an MC, coupled with the demands placed on them by the types of applications that they are intended to be used for, make reliability concerns a very important issue. We noted in Chapter 1 that one method of improving reliability is to design a system in such a way that it will be able to tolerate component failures and still remain operational. For an MC, fault-tolerance at the *system level* (also referred to as the *topological level*) implies that the type of faults to be tolerated are processor and/or link failures, and an MC (by an MC we will often mean its topology) is said to be *fault tolerant* if it can remain *functional* in the presence of such failures. It is, however, the topological requirements of the application that essentially determine when an MC is considered to be functional.

Two basic functionality criteria have received much attention. According to one of these criteria, an MC is considered functional as long as there is a nonfaulty communication path between each pair of nonfaulty processors [ArLe81, PrRe82, RaGA85, Boes86,

Esfa88]. In other words, the underlying topology of the MC should remain *connected* in the presence of certain failures. Both processor and link failures have been considered in the literature. This fault-tolerance model is particularly applicable to large-scale MCs that are to execute concurrent algorithms whose performance is mostly insensitive to infrequent changes in the topology of the system. Such MCs generally permit graceful degradation.

The second functionality criterion considers an MC functional as long as a *desired* topology is contained in the system. In the past few years much work has been done in developing parallel algorithms and the best MCs for their executions [Quin87]. For these algorithms, the existence of certain topologies is a significant factor in delivering the desired performance. Thus, for such applications, the system should be able to provide a specific topology throughout the execution of the algorithm. The existing work on such fault-tolerant topologies has mainly considered processor failures. The basic approach in achieving fault-tolerance when this functionality criterion is used is to employ system-wide redundancy. Two topologies that have received much attention are the *ring* and the *tree* [Haye76, KwTo81, RaAE84, LoFu87, DuHa88]

Here, fault-tolerance will refer to the second functionality criteria, and we will consider only processor failures. We assume that all failures are permanent and that some diagnosis mechanism is available to detect and isolate the faulty processors. Further, it is assumed that in the event of processor failures, some mechanism exists that allows the system to be "reconfigured" (using redundant processors) to maintain the desired topology. This also implies that as a part of this process, the tasks assigned to the faulty processor can be successfully shifted to a fault-free processor. Neither the diagnosis nor the reconfiguration problems will be addressed here [YaHa86].

In designing such a fault tolerant topology, the number of faults to be tolerated is specified *a priori*. Given this value, say k, a topology is determined such that the desired sub-topology can be realized given any collection of k faults. In previous work, the

primary design criteria has been to employ the minimum number of redundant processors in deference to other parameters, such as the number of communication links, etc. The overriding rationale for this choice has largely been economic: processors are usually the most expensive component. However, using the absolute minimum number of spare processors has typically forced these designs to incorporate a very large number of redundant communication links; in graph theoretic terms each vertex has very "high" degree. From a practical perspective this presents a problem in that processors have only a finite (usually quite small) number of communication ports. For large scale systems these designs can only be used to achieve low levels of fault tolerance, since for high levels, the number of ports needed would exceed the number available. Additionally, in terms of VLSI and WSI technology, one of the primary design concerns is overcoming pin-limitation and layout problems; designs which require large numbers of links only exacerbate these problems [ChCD88, Dall88].

In this chapter, we present an alternate primary design criterion and employ it in the design of fault-tolerant loop topologies. The basic idea is to place a bound on the maximum number of connections at each processor and determine the number and configuration of redundant components required to achieve a specified level of fault-tolerance. Loops are a common interconnection topology and are important in their own right. However, as many other topologies have loops "embedded" within them, results relating to the construction of "low" degree fault-tolerant loops can give insight into the feasibility of such constructions for other topologies.

4.1. Background

In this section we introduce some additional terminology, definitions and a formal statement of the problem we will consider. As we have done throughout this dissertation, we model the topology of an MC by a graph $G(,V,E)$. In particular, a loop interconnection network connecting N processors is modeled by the cycle C_N. The failure of a

processor is modeled by the removal of the corresponding vertex and its incident edges in the graph.

As previously mentioned, in this model an MC is functional as long as a desired structure is contained within the system. This criterion was first formulated by Hayes and can be stated formally as follows [Haye76]. Let $G(V,E)$ represent the topology of an MC and $D(V,E)$ be a desired structure. Then, G is *k-fault-tolerant* (k-ft) with respect to D, if for any set of faulty vertices $F \subset V(G)$, with $|F| = k$, the graph $D \subseteq G - F$. We also say that G is a k-ft D graph. Thus, G is a 1-ft N-cycle means that $C_N \subseteq G - \{v\}$ for each $v \in V(G)$. It is clear from this definition that *G is a k-ft D graph* implies $|V(G)| \geq |V(D)| + k$. For a given graph D, and specified integers k and m, the set of all k-ft D graphs G, with $|V(G)| = |V(D)| + m$, will be denoted by $\Gamma_k[D,m]$.

Using the above definition, several classes of problems can be formulated by imposing certain requirements on the graph G. The problem originally proposed by Hayes can be stated as: Given a graph D, and a positive integer k, construct an "optimal" graph, G, which is k-ft with respect to D. The optimality criteria considered in [Haye76] is:

(a) $|V(G)| = |V(D)| + k$

(b) $|E(G)|$ is as small as possible subject to (a).

In [Haye76] three candidates for the graph D are considered, namely cycles, trees and simple paths. In particular, a construction is given for G when D is a cycle of even order. In this case, G is a $(k+2)$-regular graph. A similar construction is also given for the case of D being a cycle of odd order. The work of Hayes was followed by the results in [KwTo81, RaAE84, LoFu87, DuHa88]. In these papers the authors have adhered to the same optimality criterion, but imposed additional restrictions on the set F. For example, in [RaAE84] the case of D being a binary tree is considered with the additional restriction that failures only occur at different levels of the tree.

The criterion (a) alone dictates that the number of spares be exactly equal to the desired level of fault tolerance. When no additional restriction is placed on the set F, this criterion implies $\delta(G) \geq \delta(D) + k$. Thus the degree of each vertex in G is at least proportional to the number faults to be tolerated. Since current technology limits the number of connections at each processing node (and it seems this constraint will persist for some time to come [ChCD88, DAll88],) the above optimality criteria may not be viable in certain situations.

To address the above concern, we propose the following modified version of the problem. Given a desired graph D and positive integers k and r, construct a graph G such that:

(a) D is a subgraph of $G-F$ for any set $F \subset V(G)$, with $|F| = k$.

(b) G is r-regular, and

(c) no graph satisfying (a) and (b) above has fewer vertices than G.

It is clear that this problem may not have any solution for certain choices of D, k, and r. A case in point, when $r=2$ and $D = C_N$, there is no solution G for any k. Thus 3 is the smallest value of r for which a solution may exist for the case of $D = C_N$.

In the sequel we will consider the modified problem for the case $r=3$ and $D = C_N$, N even. We will show that no solutions using exactly k spares exist; thus we must incorporate "extra" redundancy into the design. In fact, we show that the minimum number of spares required is bounded by $2k$, that is $|V(G)| \geq |V(D)| + 2k$, and we will exhibit a class of graphs, called *Chordal Rings* for which the lower bound is achievable for $k=1,2,3$. Further, this result will be shown to be "optimal" in the sense that higher levels of fault tolerance cannot be achieved for this class of graphs using the minimum number of spares.

4.2. Chordal Rings as k-ft Cycle Topologies

In this section we will define the *Chordal Ring* class of graphs, and we examine the extent to which they can serve as fault-tolerant graphs for C_N. The results are presented in the form of several theorems, organized as follows. Theorem 4-1 establishes a lower bound on the number of spares required by the use of cubic (3-regular) graphs. Theorems 4-2 and 4-3 establish that certain members of the chordal ring class are 1-ft and 2-ft for C_N. Two lemmas are given to establish Theorem 4-4, which provides a necessary condition for any graph G to be a member of the class $\Gamma_k[C_N,2k]$. This theorem is used in proving Theorem 4-5, which establishes that 3-ft is the highest level of fault-tolerance that can be achieved using Chordal Rings. Finally, in Theorem 4-6, we prove that this level of fault tolerance can be achieved by the chordal rings $CR(M,7)$. We begin with some additional definitions that we make use of in the proofs.

Definition 4.1. A *fault-set* F is a set of faulty vertices.

Definition 4.2. Given a graph G and a desired graph D, a vertex v is *unusable* with respect to a given fault set F, if no subgraph of $G - F$ isomorphic to D contains v.

Example 4.1 Let G be 3-regular and the desired graph D be a cycle. Let $\{u_1,u_2,u_3\} \subset V(G)$ and $\{(u_1,u_2),(u_2,u_3)\} \subset E(G)$. Then the fault set $\{u_1,u_3\}$ makes u_2 unusable because u_2 has only one edge connected to a non-faulty vertex in G, implying that u_2 cannot be a vertex in any cycle. It is "trapped" between two faulty vertices. Note that by this definition any faulty vertex is unusable. A vertex is *usable* if it is not unusable.

Definition 4.3. Given a graph G, a fault set F, and a desired graph D, the set of all unusable vertices induced by F will be denoted $U(F)$ or simply U.

Clearly, a necessary condition for a graph to be k-ft for C_N is that there be N usable vertices for every fault set F of order k. In other words, the maximum cardinality of an unusable set may not exceed the total number of spare vertices. As we have noted, the tradeoff in using cubic graphs for k-ft cycles is that we must include "extra" spare vertices in the design. Theorem 4-1 establishes the lower bound (cited earlier) on the

amount of redundancy required.

Theorem 4-1: Let $G \in \Gamma_k[C_N,m]$ be 3-regular and connected. Then $m \geq 2k-1$.

Proof. By definition, there is an N-cycle in G. Label the vertices of this cycle $v_0,v_1,\ldots,v_{N-1}$ and consider the fault set $F = \{v_1,v_3,\ldots,v_{2k-1}\}$. As in the example 4.1, the failure of the vertices v_{2i-1} and v_{2i+1}, $1 \leq i \leq k-1$, "traps" the vertex v_{2i}, making it unusable. Thus all the vertices "between" such pairs of vertices in F are unusable. This gives $\{v_1,v_2,\ldots,v_{2k-1}\} \subset U$ and hence $|\ U\ | \geq 2k-1$. ■

Since we are considering designs for even N, Theorem 4-1 and the fact that there exists no cubic graph of odd order imply that the minimum number of spares is at least $2k$. Hereafter we restrict our attention to cubic graphs having this minimum number of spares, i.e., those in the class $\Gamma_k[C_N,2k]$. In addition, we will consider only the cases where $k < N/2$. This seems a reasonable requirement for large scale systems and eliminates many special cases in the proofs.

4.2.1. Chordal rings

A chordal ring of degree 3, hereafter referred to simply as a chordal ring, is a 3-regular Hamiltonian graph. Clearly, chordal rings are of even order. Following [ArLe81] we define a specific class of chordal rings, $CR(N,w)$, which can be constructed using Procedure 4.1. The parameter w is called the *chord length* which is required to be odd and at least 3. Note that $CR(N,w)$ is *bipartite* with v_i and v_{i+1} being in different partitions. Without loss of generality, we will assume that $w \leq N/2$. The chordal ring $CR(20,3)$ is shown in Figure 4-1. (For sake of readability, we have labeled the graphs in the figures using integral labels, i.e., vertex v_i is labeled simply i.)

Procedure 4.1. Construct $CR(M,w)$

1. Construct a cycle of order M; labeling the vertices $0,1,\cdots,M-1$.
2. For each i, $0 \leq i < (M-1)/2$, add the edge $(2i,j)$ where $j = (2i + w)\ mod\ M$.

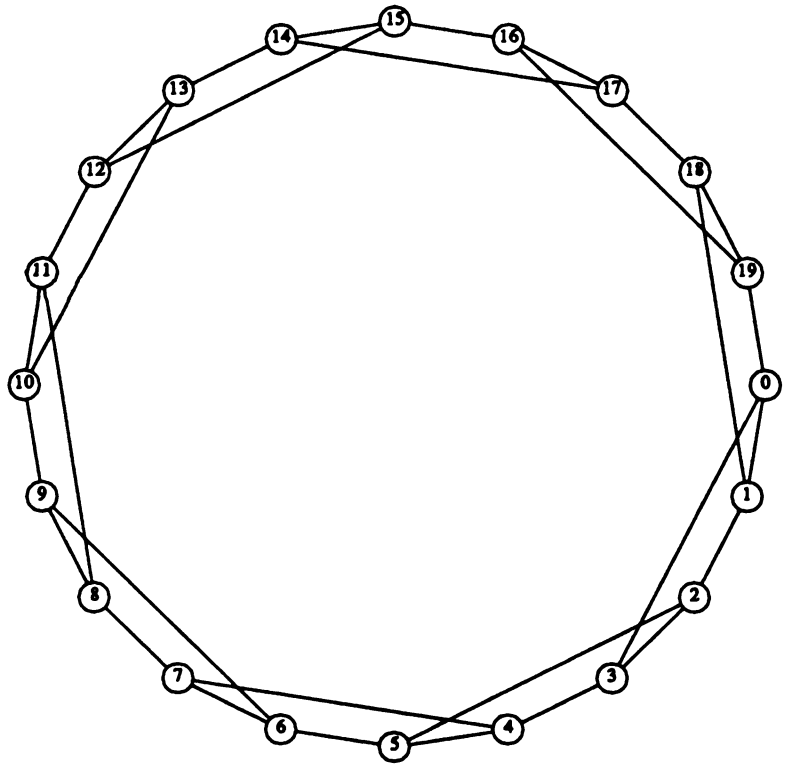

Figure 4-1. The Chordal Ring CR(20,3).

4.2.2. Fault tolerance of CR(N,w)

As we noted earlier, two solutions for 1-ft cycles were given in [Haye76]. The solution for even cycles has $\Delta = 4$, while the solution for odd cycles is cubic. We have noticed that Hayes' solution for 1-ft C_{2j+1} (using one spare) can also serve as a 1-ft C_{2j} (using two spares). In fact Hayes' design is the chordal ring $CR(2j+2,3)$. Thus such chordal rings belong to $\Gamma_1[C_{2j},2]$. The following theorem establishes a stronger result for 1-ft cycles.

Theorem 4-2: For any even $N \geq 4$, $CR(N+2,w) \in \Gamma_1[C_N,2]$, where w is as defined above.

Proof. Let N be given and let the vertex set of $CR(N+2,w)$ be $V = \{v_0,v_1,\cdots,v_{N+1}\}$. Without loss of generality, let $F = \{v_0\}$. Then it is not difficult to see that the N-cycle

$$v_1, v_2, v_3, \cdots, v_{N+1-w}, v_{N+1}, v_N, \cdots, v_{N+3-w}, v_1$$

is contained in $CR(N+2,w) - F$. ■

For the 2-ft case, we have proved [ZiEs89] that no cubic graph exists for $N=4$, and we thus consider $N \geq 6$. The next theorem establishes that for such N, the chordal rings $CR(N+4,3)$ are 2-ft for C_N.

Theorem 4-3: For any even $N \geq 6$, $CR(N+4,3) \in \Gamma_2[C_N,4]$.

Proof. Let N be given and let the vertex set of $CR(N+4,3)$ be $V = \{v_0, v_1, \cdots, v_{N+3}\}$. To establish the result, we must show that for any fault set $F = \{x, y\} \subset V$, $C_N \subseteq CR(N+4,3) - F$. Since the design uses 4 spare vertices, for any fault set F as above, two additional vertices will not be used in the resulting cycle. The two faulty vertices and the two additional ones will be referred to as the *inactive* vertices for the fault F. Without loss of generality, we let $x = v_0$ and consider the possibilities for y. There are 3 cases:

Case 1. $y = v_1$ Consider the two paths:

$$v_3, v_4, v_7, v_8, \cdots, v_{4i-1}, v_{4i}, \cdots, v_{N-1}, v_N, v_{N+1}$$

$$v_2, v_5, v_6, v_9, \cdots, v_{4i-2}, v_{4i+1}, \cdots, v_{N-3}, v_{N-2}, v_{N+1} \ .$$

These two paths have only the vertex v_{N+1} in common. A cycle of order N can be formed by concatenating the two paths and including the edge between v_2 and v_3. Note that the vertices v_{N+2} and v_{N+3} are inactive in this cycle. An example of such a cycle is given in Figure 4-2 for CR(20,3). The edges in the 16-cycle are shown as solid lines; dashed lines represent unused edges.

In the remaining two cases we will exploit the Hamiltonian cycle shown in Figure 4-3 to establish the result. This cycle is made up of paths of the form $v_{2i+1}, v_{2i}, v_{2i+3}$; that is, an edge from the "peripheral" cycle followed by a chord in the "opposite" direction. It is clear that such a cycle always exists in CR(M,3). The salient feature of this Hamiltonian cycle is that we may remove any two "endpoints" of a chord and using an edge

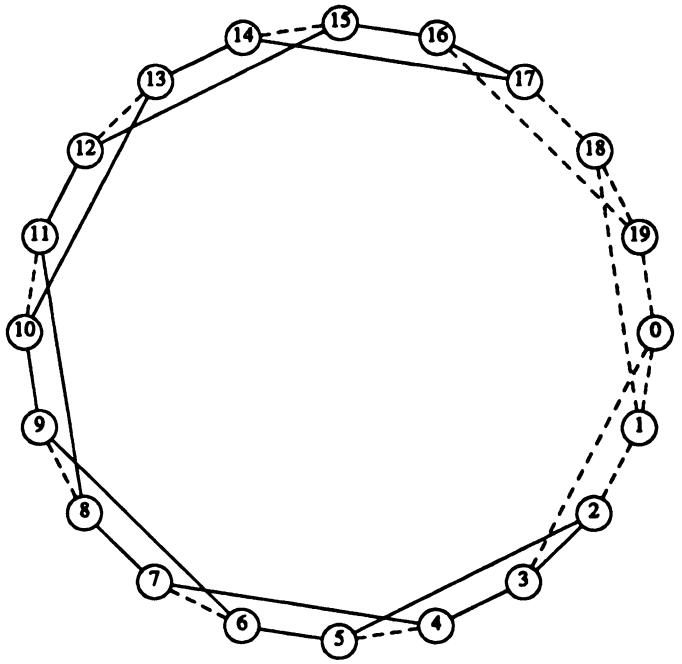

Figure 4-2. A 16-cycle in $CR(20,3) - \{0,1\}$.

previously not used, obtain a new cycle *without altering any other edge or vertex* in the original cycle. The order of the new cycle is obviously two less than that of the initial one. For example, we may remove the vertices v_0 and v_3 in Figure 4-3 and employ the edge (v_1,v_2) to obtain the cycle shown in Figure 4-4.

Case 2. $y = v_j$, $j \neq 1,3$ In this case, an N-cycle can be obtained by removing the chords (i.e., their endpoints) corresponding to the faulty vertices as discussed above.

Case 3. $y = v_3$. Here, the two faulty vertices are the endpoints of a chord. To obtain a cycle of the correct size, we choose the endpoints of any other chord in the graph (with the exception of the pair v_1, v_{N-3}) and remove them to obtain a cycle of the desired size as in case 2.

Thus for all combinations of 2 failures we can obtain a cycle of order N. ■

We have shown that $CR(N+2k,w) \in \Gamma_k[C_N,2k]$ for $k=1,2$ and $w=3$. We will next show that the 3-ft is the best that can be achieved from chordal rings using the minimum number of spare vertices. To do this we need the following results.

Lemma 4-1 : Let G be a 3-regular and connected graph. Further, let $G \in \Gamma_k[C_N,2k]$. Then the girth of G, $g(G) \geq k+1$.

Proof. By contradiction. Suppose G has a j-cycle C_j, $j \leq k$. Consider the set $A = \{v \mid v \notin C_j, v$ is adjacent to some vertex $u \in C_j\}$. Call A the *adjacent set* of C_j and let $|A| = m$. Note that $m \leq j$, since each vertex in C_j has one incident edge not in the cycle with which to connect to another vertex in G. Suppose all the vertices in A are faulty. The vertices in C_j are "trapped" by the fault set A and thus $(A \cup C_j) \subset U$. Since G is k-ft for C_N and $m \leq k$, there is an N-cycle in G which does not include any vertices in $A \cup C_j$. Let $w_0,w_1,\ldots,w_{N-1},w_0$ be such a cycle. Since G is connected, there is a path between some vertex $v_0 \in A$ to a vertex w_0 in the N-cycle which does not include any other vertex in A. Let this path be $P = v_0,u_1,u_2,\ldots,u_p,w_0$. Note that the path P may be of length 1. Now extend P to P' by concatenating with it the path formed by the vertices of the N-cycle: $P' = v_0,u_1,u_2,\ldots,u_p,w_0,w_1,\ldots,w_{N-1}$. So, we have established that there is a path beginning at v_0, of length at least N, which does not contain any vertices of $A \cup C_j$ other than v_0. Let us relabel P', denoting the first $2k-2m+2$ vertices as $v_0,v_1,v_2,\ldots,v_{2k-2m+1}$. Now consider the fault set:

$$F = A \cup \{v_1,v_3,\ldots,v_{2k-2m+1}\} - \{v_0\}$$

which contains $|F| = m + (k-m+1) - 1 = k$ nodes. This makes the set

$$\left[A \cup C_j \cup \{v_1,v_2,\ldots,v_{2k-2m+1}\}\right] \subset U$$

and thus $|U| \geq m + j + 2k-2m+1 \geq 2k+1$, which contradicts the definition of G. ■

Lemma 4-2: Let G be as in Lemma 4.1, then $g(G) \neq k+1$.

Proof. By contradiction. Suppose G has a $k+1$-cycle C, and let A be the adjacent set of this cycle as in Lemma 4.1. If $|A| \leq k$, then the proof of Lemma 4.1 also holds here.

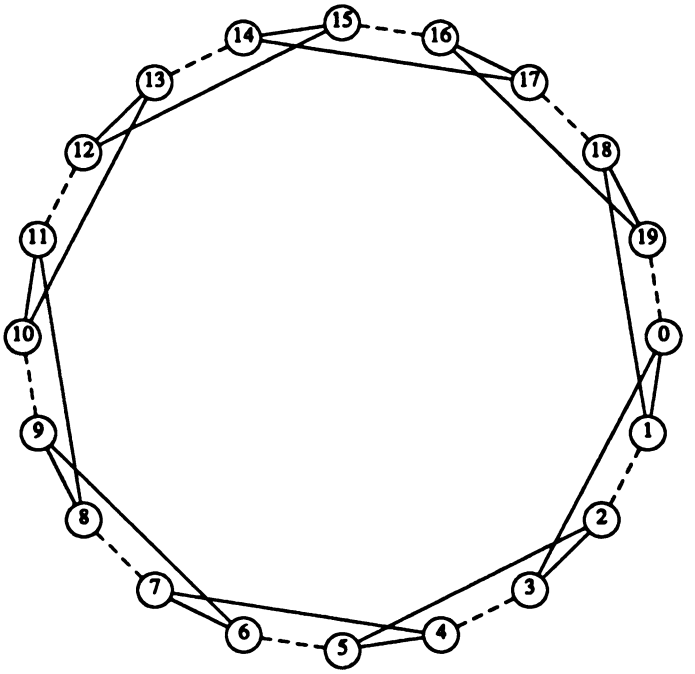

Figure 4-3. A Hamiltonian cycle in CR(20,3).

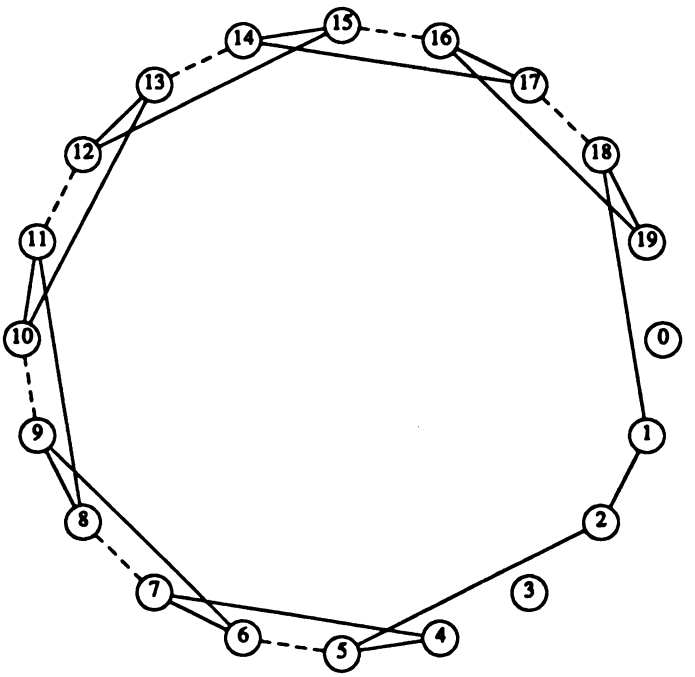

Figure 4-4. An 18-cycle in *CR* (20,3) – {0,3}

and $(C \cup F) \subset U$. This implies $|U| \geq k+1+k = 2k+1$ which contradicts the definition of G. ■

Lemmas 4.1 and 4.2 give us the following theorem.

Theorem 4-4: Let G be a 3-regular and connected graph. Further, let $G \in \Gamma_k[C_N, 2k]$. Then $g(G) \geq k+2$. ■

We are now ready to state the following theorem which indicates that 3-ft is the highest level of fault-tolerance that can be attained with Chordal Rings using the minimum number of spare vertices..

Theorem 4-5: $CR(N+2k, w) \notin \Gamma_k[C_N, 2k]$ for any $k \geq 4$.

Proof. For any values of N and w as defined above, the graph $CR(N, w)$ contains the cycle induced by the edge set:

$$\{(v_0, v_1), (v_1, v_2), (v_2, v_{w+2}), (v_{w+2}, v_{w+1}), (v_{w+1}, v_w), (v_w, v_0)\}.$$

This implies that $g(CR(M, w)) \leq 6$ and therefore from Theorem 4 the conclusion holds for $k \geq 5$. For $k = 4$, consider the fault set

$$F = \{v_0, v_2, v_4, v_{N+2k-w+2}\}.$$

This makes the set

$$F' = F \cup \{v_1, v_3, v_{N+2k-w+1}, v_{N+2k-w+3}\} \subset U.$$

Since five of these vertices all belong to one set of a bipartition of $CR(N+2k, w)$, the largest cycle that can exist in $CR(N+2k, w) - F'$ is $C_{N+2k-10}$. This gives the desired result for $k = 4$. ■

The final theorem of this chapter, establishes that the chordal rings $CR(N+6, 7)$ are 3-ft for C_N, $N \geq 20$. These graphs have chord length equal to seven. Note that since $g(CR(M, 3)) = 4$, we may conclude from theorem 4-4 that $CR(N+6, 3) \notin \Gamma_3[C_N, 6]$ for any N. In addition, for the chordal rings $CR(N+6, 5)$, we have found counterexamples which show that the graphs are not member of $\Gamma_3[C_N, 6]$ for arbitrarily large values of N.

Theorem 4-6: $CR(N,7) \in \Gamma_3[C_M,6]$, where $M=N-6$ and $N \geq 26$.

Proof. We need to show that for any fault set $F = \{x,y,z\}$, $C_M \subseteq CR(N,7)-F$.

We proceed by specifying fault-sets F, and demonstrating that an M-cycle is present in the presence of each such fault-set. For some fault sets, a specific cycle will be given explicitly. In the remaining cases, the existence of an M-cycle in a specific graph or set of graphs along with an inductive argument will demonstrate that the fault-set can be tolerated for every $CR(N,7)$, where N is as in the statment of the theorem. The proof employs four types of inductive arguments, which are illustrated in Figures 4-5, 4-6, 4-7, 4-8. A fifth type of argument (non-inductive) shows how a special *class* of faults can be handled. Each of these arguments is described below. In our approach, each fault set is characterized in terms of the relative distances between its member vertices; the inductive argument is always done in such a way as to preserve these relative distances, and hence the fault-set itself.

Figure 4-5 illustrates the most straightforward inductive variation. In the figure, all the faults are contained "within" one chord. The M-cycle is apparent. It is clear that we may add a pair of vertices to this graph, (anywhere except between vertices 26 and 5) extending the cycle by two, and not altering the fault set. We thus obtain a chordal ring of order $N+2$, which has an $M+2$ cycle for the given fault-set. The key characteristic here is that there be an edge in the M-cycle of the form $(2i+1,2i+2)$ such that there is no chord in the M-cycle which begins at the vertices $2i-4$, $2i-2$, $2i$. Intuitively, no edge of the M-cycle "jumps" past the vertices $2i+1$, $2i+2$. It is clear that in such a circumstance, we may add any even number of vertices between vertices $2i+1$ and $2i+2$, which will establish the pattern for all chordal rings. We will refer to this as the **induct-by-two (IB2)** pattern.

The second inductive variation is show in Figure 4-6. The key characteristic is that there be an edge in the M-cycle of the form $(2i,2i+7)$ (i.e. a chord) and that the edges $(2i+2,2i+9)$ and $(2i-2,2i+5)$ are not used in the M-cycle. Intuitively, imagine drawing

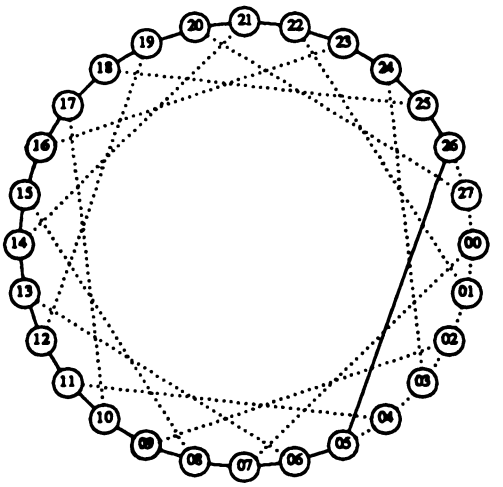

Figure 4-5. An induction-by-two illustration.

a straight line from the center of the circle, between two vertices such that the only edge of the M-cycle intersected is a chord. Observe that such a cycle can be extended by adding four vertices between vertices $2i+3$ and $2i+4$, as shown in the bottom graph. In the example shown, $2i = 10$ and the four new vertices are labeled "N" for "new". To make use of this pattern, we will demonstrate the desired cycle in graphs of orders N and $N+2$, and then use this property to induct by four to yield the conclusion. We will refer to this as the **induct-by-four (IB4)** pattern.

Figure 4-7 illustrates the first of two "induct-by-six" patterns. The key feature in this first pattern is that the M-cycle "doubles-back" on itself. Intuitively, a line can be drawn which does not intersect any edge of the M-cycle. Note that "doubling back" implies the existence of two "corners": vertices 5 and 6 are corners in the upper graph. Observe that any such cycle can be extended to a cycle with 6 additional vertices as shown in the lower graph. The extension can be accomplished by removing the peripheral edge (6,7), adding 6 new vertices between vertices 5 and 6 and reforming the

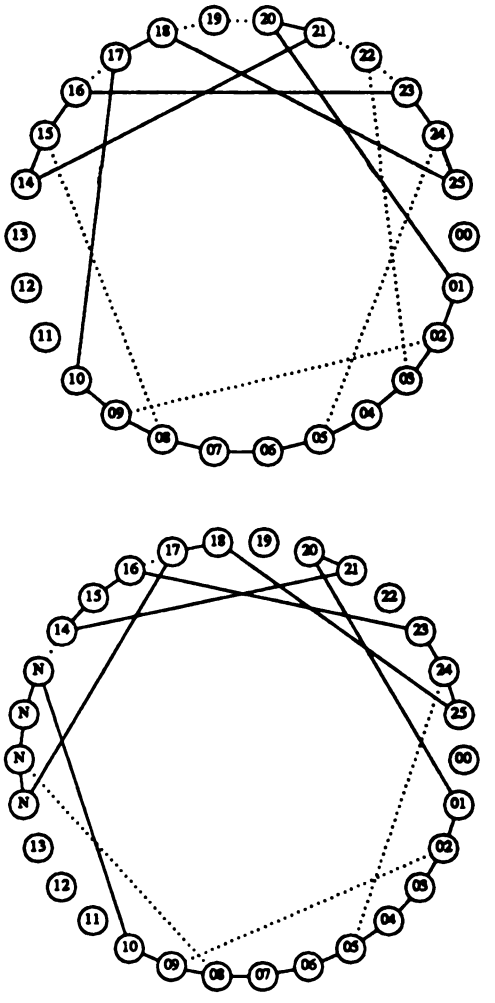

Figure 4-6. An induction-by-four illustration.

cycle as shown. Note that the extension can be done at either corner. Since this process adds six vertices, to use it inductively it will be necessary to establish that a cycle exists for a given fault-set in graphs with orders $N,N+2,N+4$. We will refer to this as the **induct-by-six-A (IB6A)** pattern, Figure 4-7.

The fourth variation will be referred to as the **induct-by-six-B (IB6B)** pattern, illustrated in Figure 4-8. The characteristic here is that the M-cycle uses three consecutive chords $(2i,2i+7)$, $(2i+2,2i+9)$ and $(2i+4,2i+13)$ and does not use the edge $(2i+5,2i+6)$. The cycle is extended by adding six vertices between $2i+5$, $2i+6$ as shown ($2i$ = 8 in the example). Again, this situation may be intuitively characterized as being able to draw a line from the center of the circle between a pair of vertices $(2j+1,2j)$ which intersect exactly three edges of the cycle, all of which are chords.

The final type of argument we use is illustrated in Figure 4-9. The idea is to identify a cycle pattern for a specific fault-set and observe that the fault set can be "pushed" forward and the cycle reformed. In the top graph, the fault is F = {0,3,4,5}. The cycle uses the edge (1,2) and then "skips over" vertices 3,4,5 using a chord. The cycle then traverses groups of four vertices connected by peripheral edges, with each group connected to the next by a chord. Observe in the middle graph that the fault-set has been advanced forward by four vertices, i.e. F = {0,7,8,9}. and the cycle reformed. Essentially, one of the groups of four vertices has been displaced (backwards) by the advanced fault set. This process can be continued until all the groups of four have been displaced, as seen in the bottom graph. Note that typically, as in the figure, the cycles we exhibit have specific patterns before and after the fault-set being advanced. The number of vertices needed by these patterns (before and after) varies between fault-classes, but typically meets a certain congruence relation and range criteria. For example, the fault class above can be described as follows. F = {0,$2i-1$,$2i$,$2i+1$}, where $4 \le 2i \le N-14$ and $N-2i \equiv 2 \; mod \; 4$. This argument will sometimes be used in conjunction with the four inductive arguments in establishing the result for certain fault-sets. We will refer to this

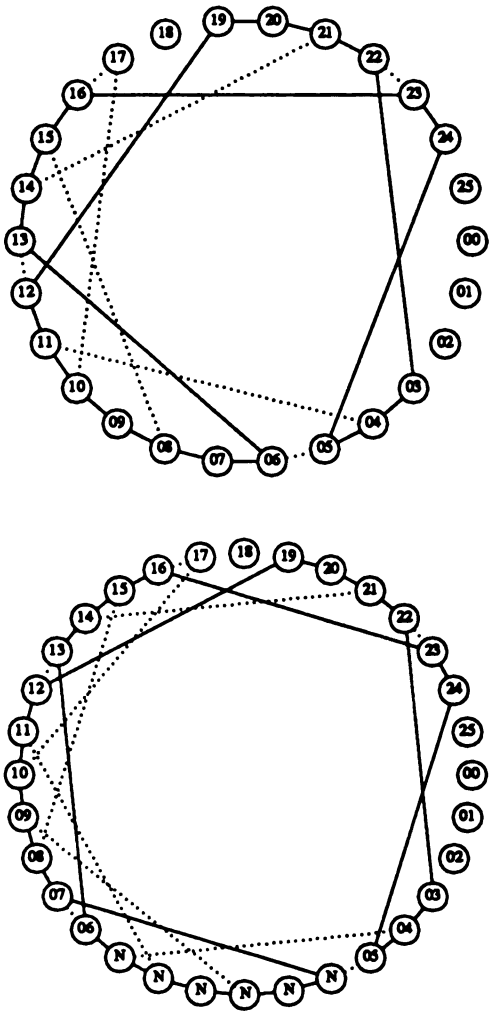

Figure 4-7. An induction-by-six, version A illustration.

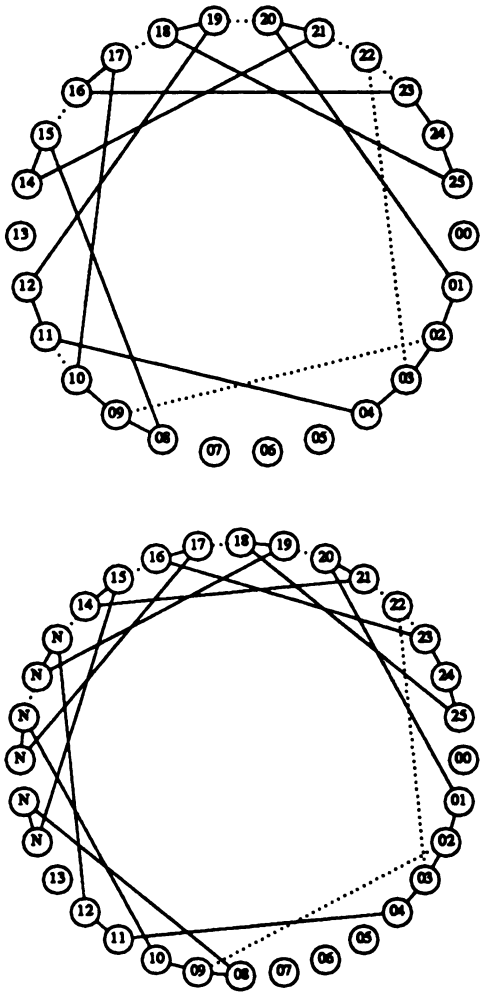

Figure 4-8. An induction-by-six, version B illustration.

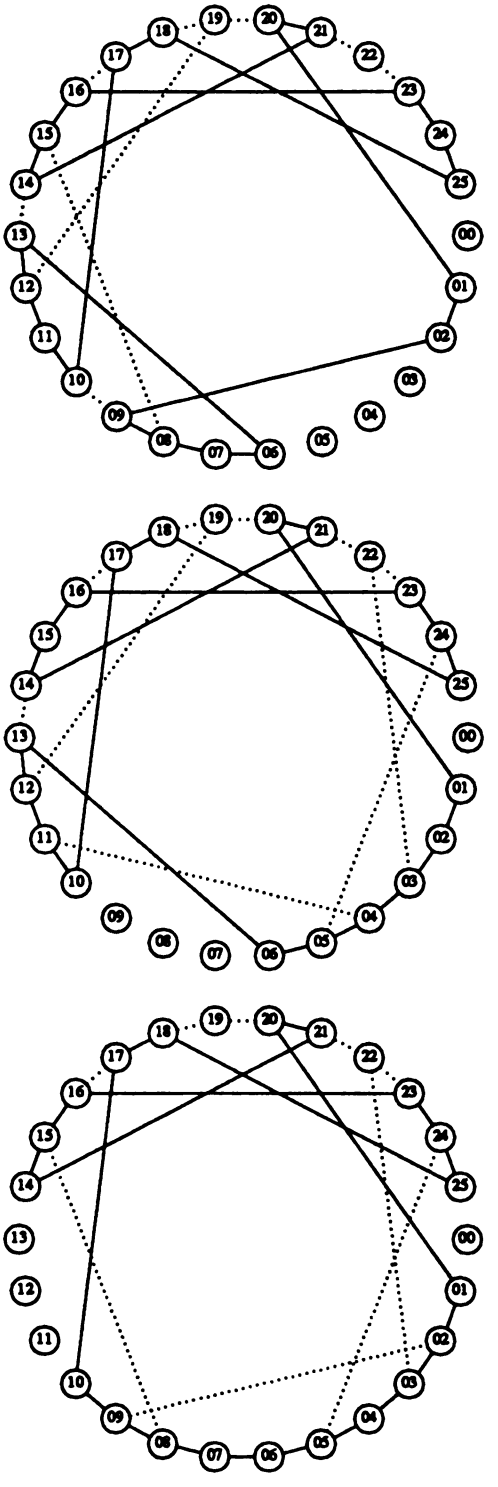

Figure 4-9. An advance-by-four illustration.

as the **advance-by-four (AB4)** pattern.

Since the graphs we are considering are bipartite, we may assume without loss of generality that $x = 0$ and then consider the possible values for y and z. Note that when y is odd, by symmetry we need only consider values of z in the range $2y \le z \le (N+y-1)/2$. Also, when y is even, we need only consider values of z where $2y \le z \le N-2y-1$.

We proceed by considering several cases based on choices for y. For each case, we provide a series of specific fault sets, numbered for convenience.

CASE 1. $y \in \{1,2\}$.

Fault Set: 1 $F = \{0,1,2,3,4,N-1\}$. Here, all the faults are contained "inside" a chord, which can be used to "short circuit" the faulty vertices. See Figure 4-5. This also covers the fault $F = \{0,1,2,N-3,N-2,N-1\}$.

Fault Set: 2 $F = \{0,1,2,2i-1,2i,2i-1\}$, where $N-2i \equiv 2 \pmod 4$ and $6 \le 2i \le N-6$. See Figure 4-10. We may apply AB4. The requirement $N-2i \equiv 2 \pmod 4$ simply guarantees that the number of vertices after the fault is a multiple of four so that the group of four pattern will work.

Fault Set: 3 $F = \{0,1,2,2i-1,2i,2i-1\}$, where $N-2i \equiv 0 \pmod 4$ and $6 \le N-2i \le N-20$. See Figure 4-11. We may apply AB4. Here the path after the fault-set requires 18 vertices. The inequality aboves ensures that there are enough vertices for this pattern.

The above fault-sets cover all the odd vertices and all even vertices except $N-16,N-12,N-8,N-4$, which are considered next.

Fault Set: 4 $F = \{0,1,2,N-16\}$. See Figure 4-12. Apply IB2 to the edge (3,4).

Fault Set: 5 $F = \{0,1,2,N-12\}$. See Figure 4-13. Apply IB2 to the edge (3,4).

Fault Set: 6 $F = \{0,1,2,N-8\}$. See Figure 4-14. Note that the cycle doubles back at vertices 5 and 6. We may apply IB6A to either corner. It is important to note that we are not altering the fault pattern, since in this case we may consider the faults to be

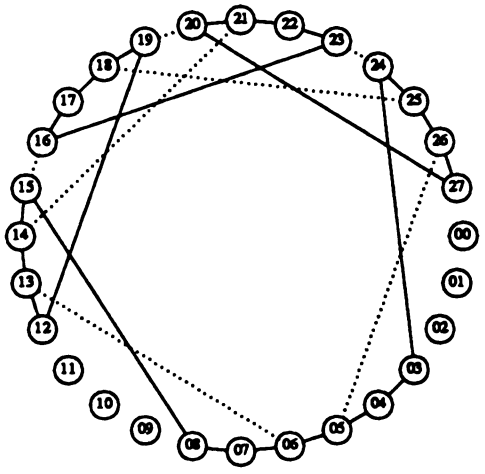

Figure 4-10. $F = \{0,1,2,2i-1,2i,2i+1\}$, $N-2i \equiv 2 \ mod\ 4$

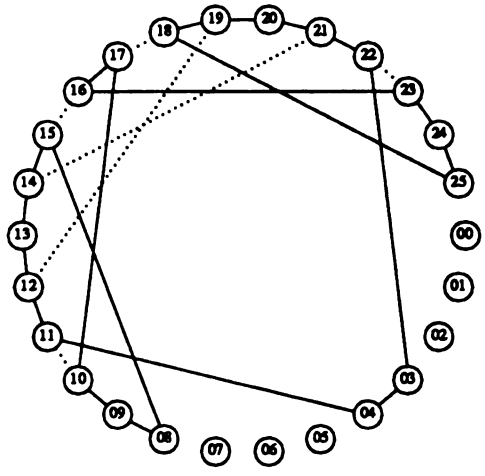

Figure 4-11. $F = \{0,1,2,2i-1,2i,2i+1\}$, $N-2i \equiv 0 \ mod\ 4$

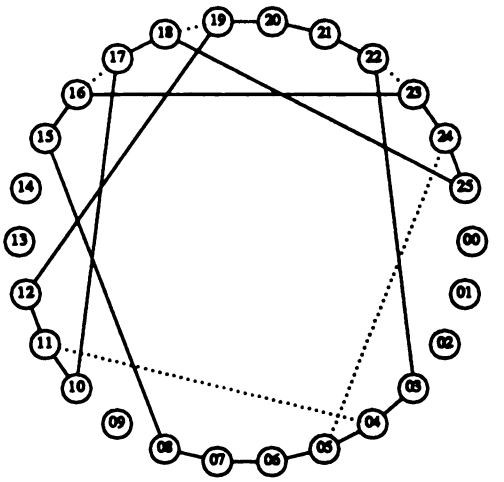

Figure 4-12. $F = \{0,1,2,N-16\}$.

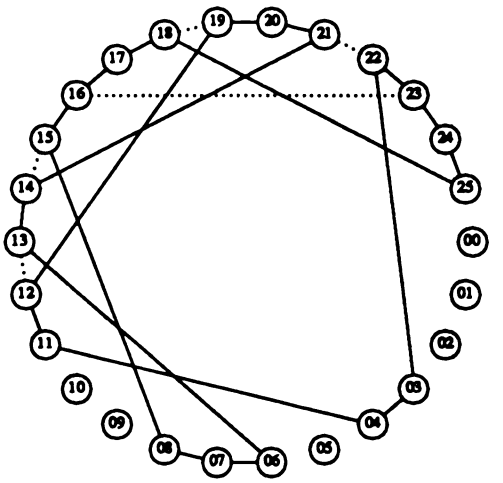

Figure 4-13. $F = \{0,1,2,N-12\}$.

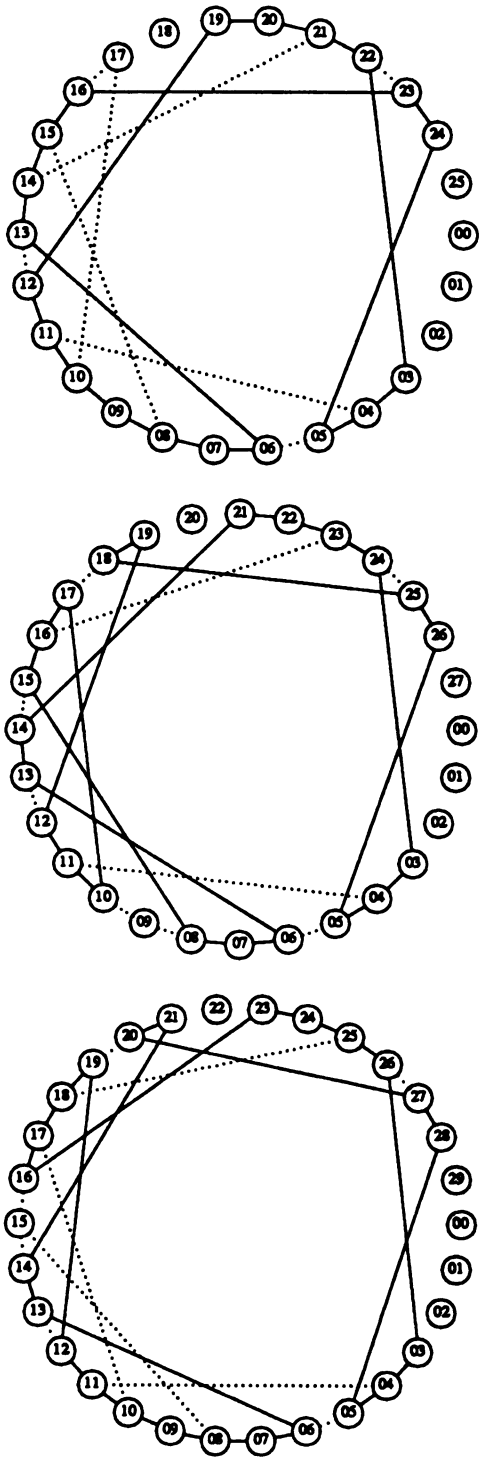

Figure 4-14. $F = \{0,1,2,N-8\}$.

"contained" between $N-8$ and 2 (in a clockwise sense from $N-8$) and altering the graph outside of this containment preserves the relative distance between faults.

Fault Set: 7 $F = \{0,1,2,N-4\}$. See Figure 4-15. Apply IB6A to either corner.

We have covered all possible fault-sets of the form $\{0,1,2,z\}$, completing this case.

CASE 2. $y = 3, 6 \leq z \leq N/2+1$.

Fault Set: 8 $F = \{0,3,6,7,8\}$. See Figure 4-16. Apply IB6A to either corner.

Fault Set: 9 $F = \{0,1,2,3,9\}$. See Figure 4-17. Apply IB6A to vertex $N-1$.

Fault Set: 10 $F = \{0,3,4,10,11\}$. See Figure 4-18. Apply IB6A to vertex 12.

Fault Set: 11 $F = \{0,3,12,13,14\}$. See Figure 4-19. Apply IB2 to the edge (15,16).

Fault Set: 12 $F = \{0,3,10,2i-1,2i,2i+1\}$. Where $14 \leq 2i \leq N-10$ and $N-2i \equiv 2 \; mod \; 4$. See Figure 4-20. Apply AB4.

Fault Set: 13 $= \{0,3,10,2i-1,2i,2i+1\}$. Where $14 \leq 2i \leq N-16$ and $N-2i \equiv 0 \; mod \; 4$. See Figure 4-21. Apply AB4.

This completes case 2.

CASE 3. $y = 4, 8 \leq z \leq N-3$.

Fault Set: 14 $F = \{0,4,2i-1,2i,2i+1\}$. Where $8 \leq 2i \leq N-10$ and $N-2i \equiv 2 \; mod \; 4$. See Figure 4-22. Apply AB4.

Fault Set: 15 $F = \{0,4,2i-1,2i,2i+1\}$. Where $8 \leq 2i \leq N-12$ and $N-2i \equiv 0 mod \; 4$. See Figure 4-23. Apply AB4.

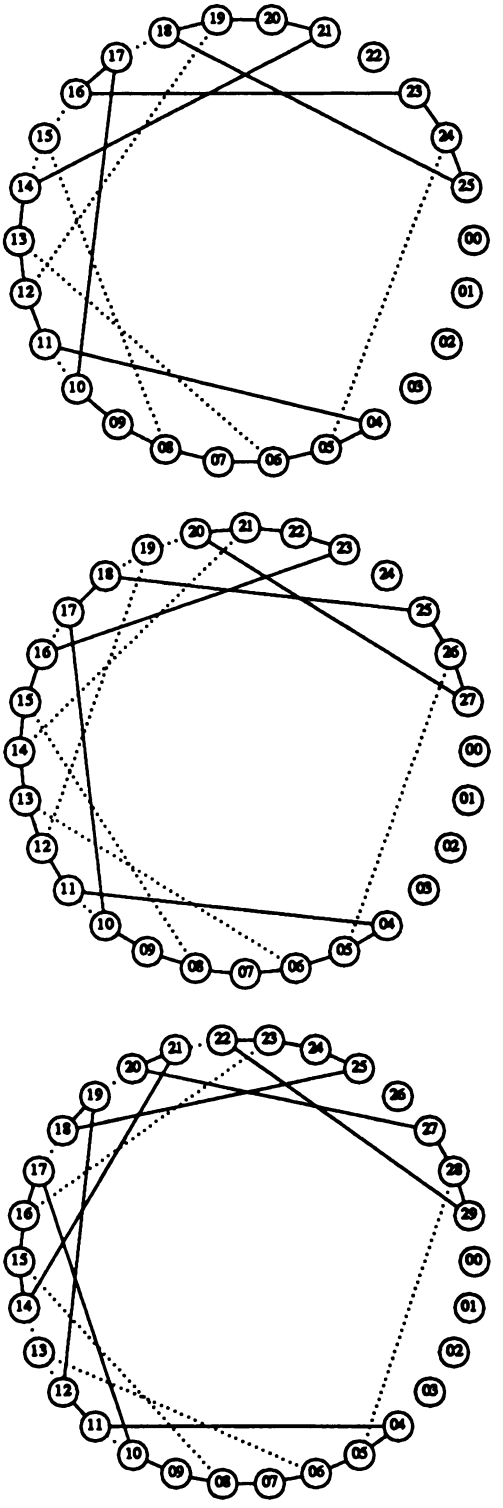

Figure 4-15. $F = \{0,1,2,N-4\}$.

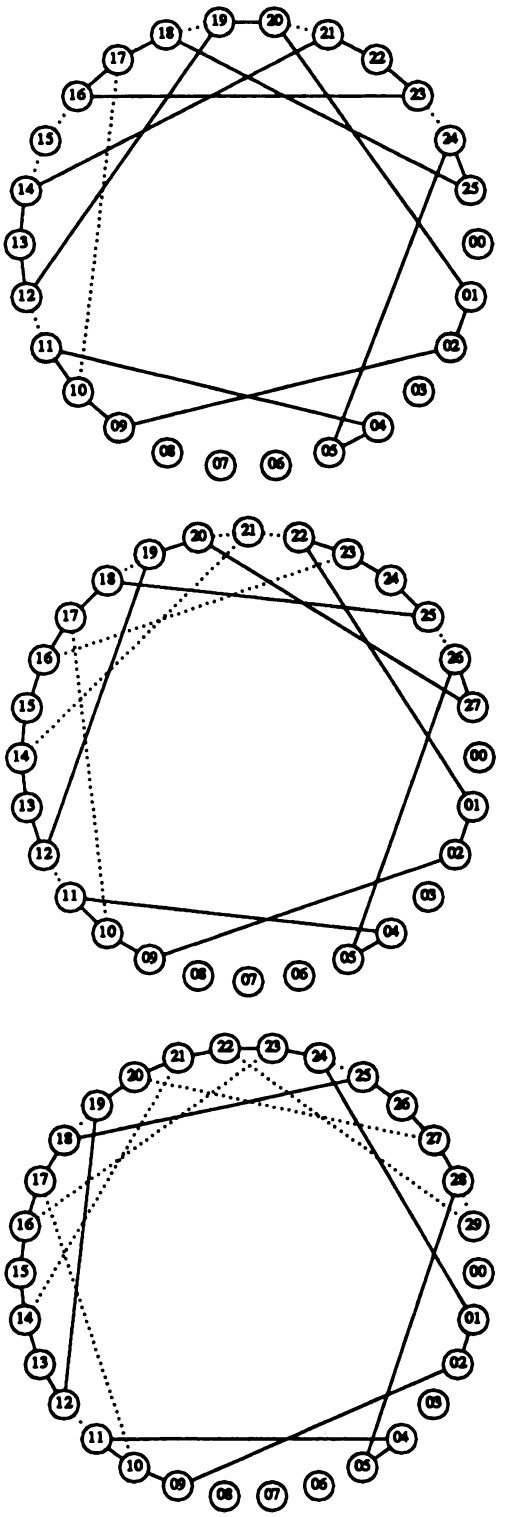

Figure 4-16. $F = \{0,3,6,7,8\}$.

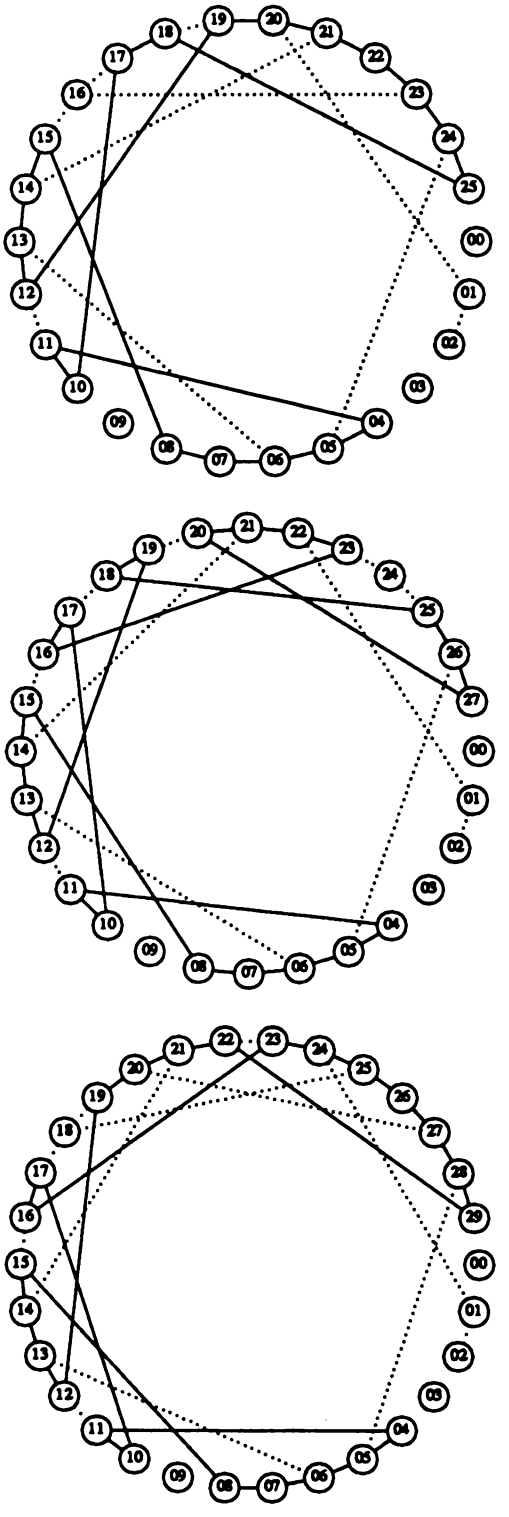

Figure 4-17. F = {0,1,2,3,9}.

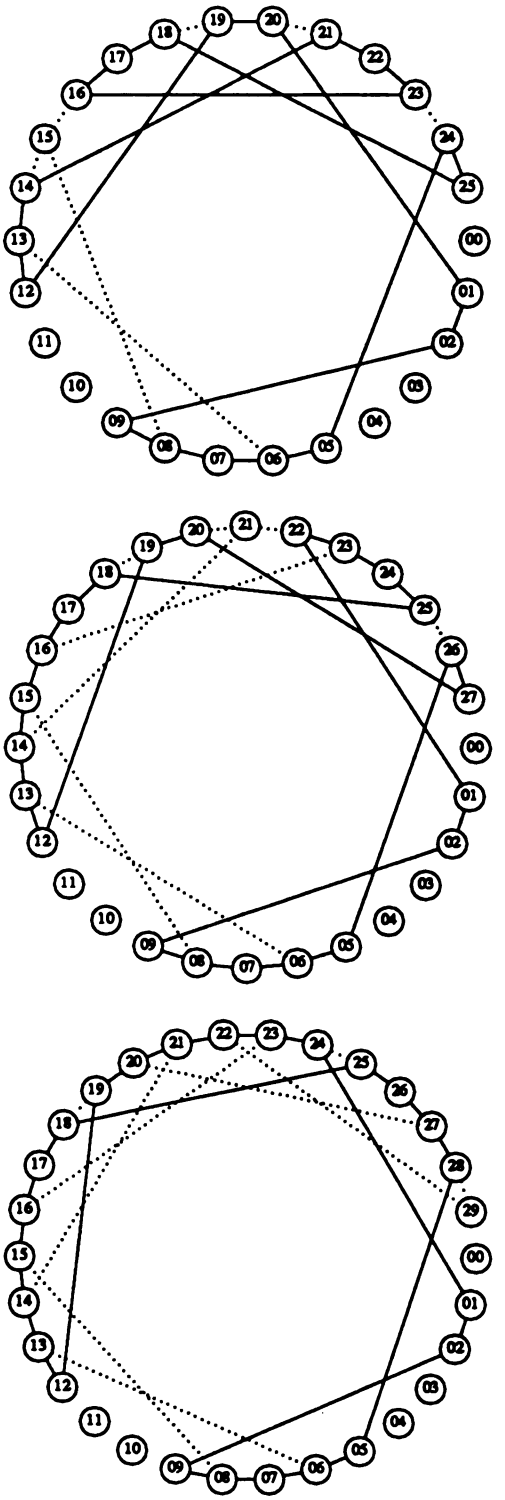

Figure 4-18. F = {0,3,4,10,11}.

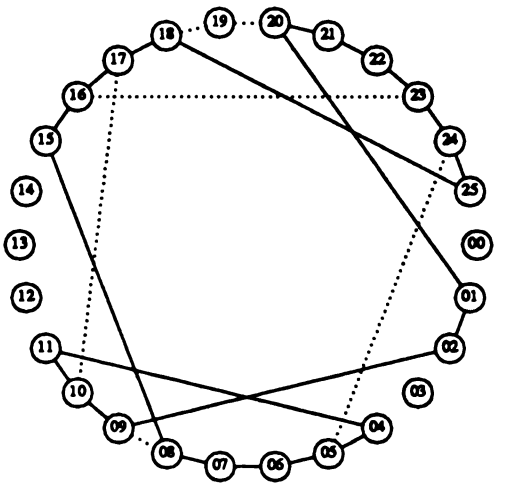

Figure 4-19. $F = \{0,3,12,13,14\}$.

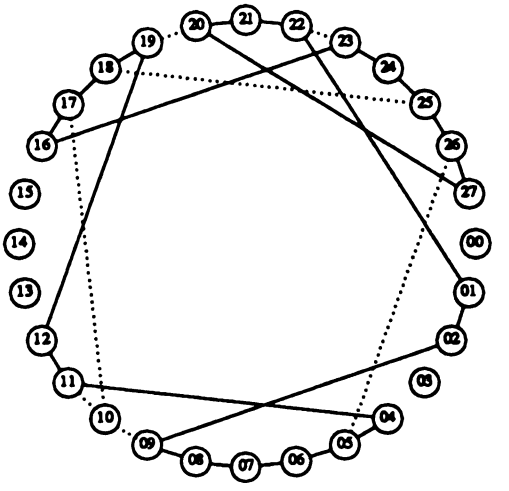

Figure 4-20. $F = \{0,3,10,2i-1,2i,2i+1\}, N-2i \equiv 2 \, mod \, 4.$

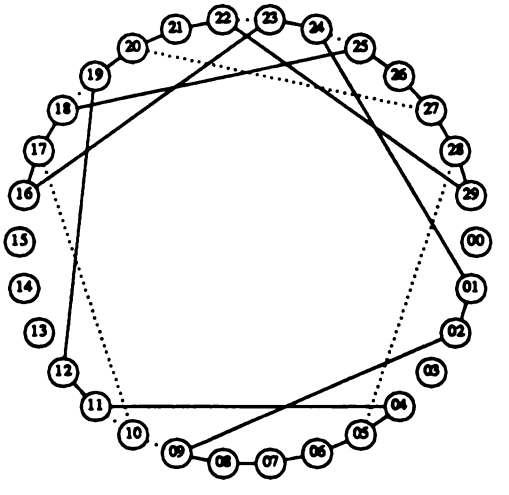

Figure 4-21. $F = \{0,3,10,2i-1,2i,2i+1\}, N-2i \equiv 0 \, mod \, 4.$

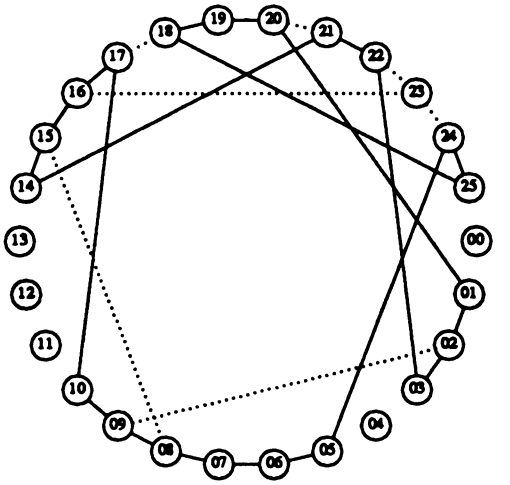

Figure 4-22. $F = \{0,4,2i-1,2i,2i+1\}, N-2i \equiv 2 \, mod \, 4.$

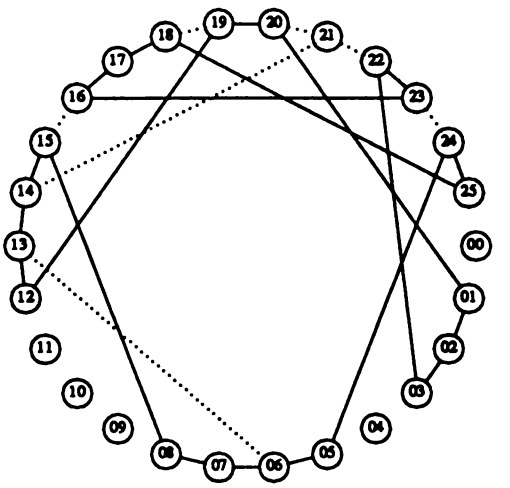

Figure 4-23. $F = \{0,4,2i-1,2i,2i+1\}, N-2i \equiv 0 \, mod \, 4.$

The previous sets cover all possible values of z in the range $8 \leq z \leq N-8$. The next three fault-sets complete this case.

Fault Set: 16 $F = \{0,4,N-8,N-7\}$. See Figure 4-24. We may apply IB2 to the edge (5,6).

Fault Set: 17 $F = \{0,4,N-6,N-5\}$. See Figure 4-25. Apply IB6A to either corner.

Fault Set: 18 $F = \{0,4,N-4,N-3\}$. See Figure 4-26. Apply IB6A to either corner.

CASE 4. $y = 5, 10 \leq z \leq N/2+2$.

Fault Set: 19 $F = \{0,5,10,11\}$. See Figure 4-27. Apply IB4 between vertices $N-1$ and 0.

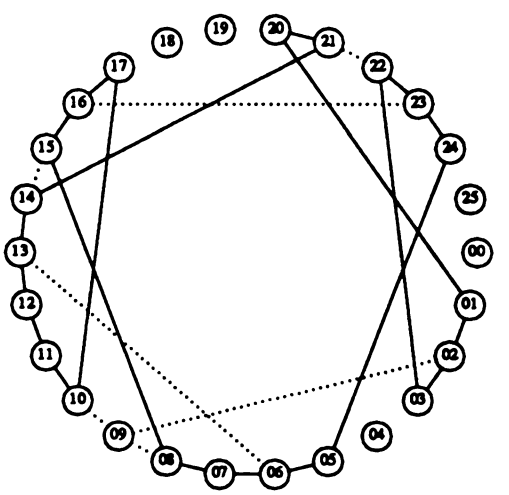

Figure 4-24. $F = \{0,4,N-8,N-7\}$.

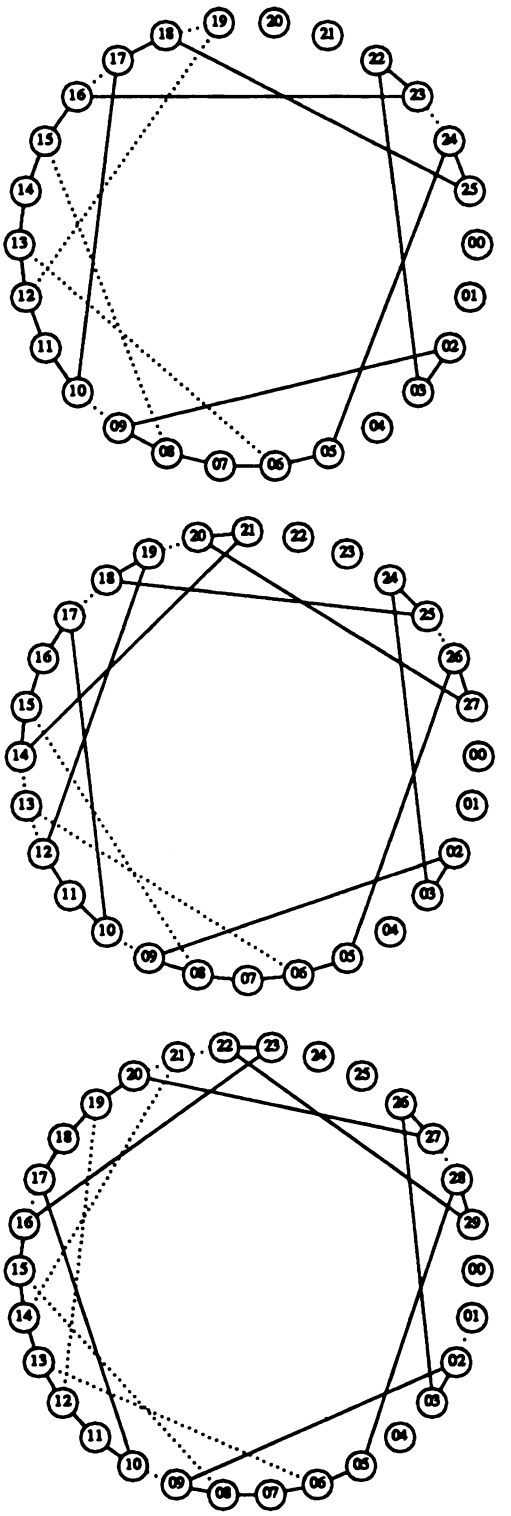

Figure 4-25. $F = \{0,4,N-6,N-5\}$.

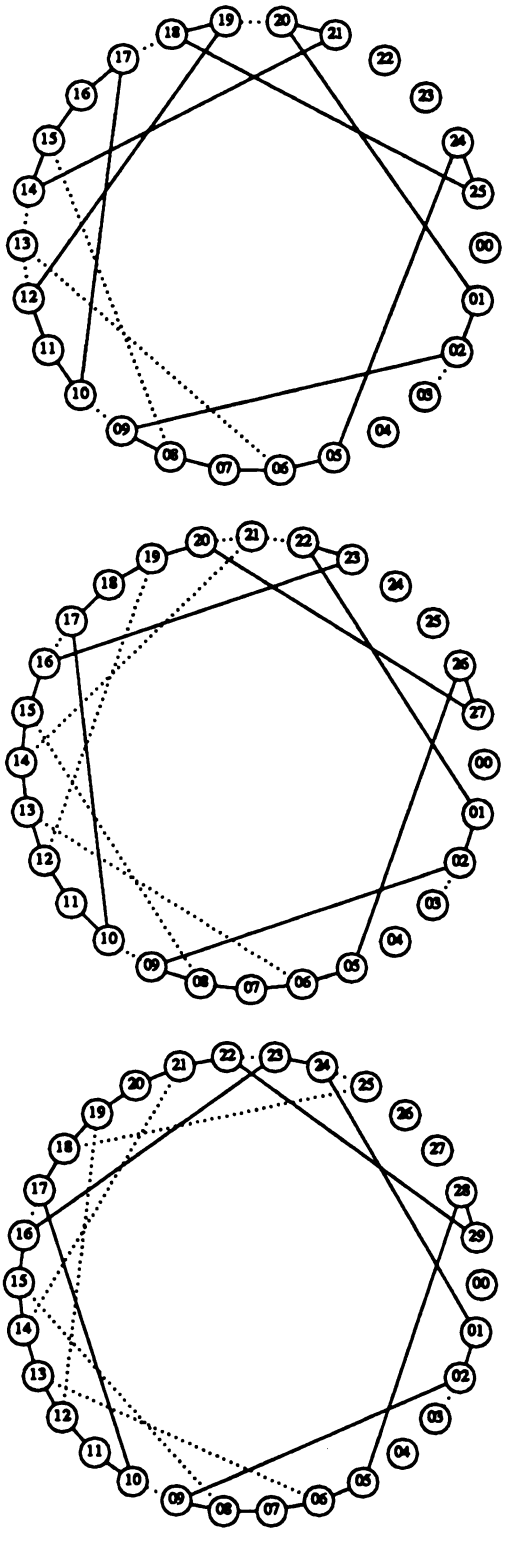

Figure 4-26. $F = \{0,4,N-4,N-3\}$.

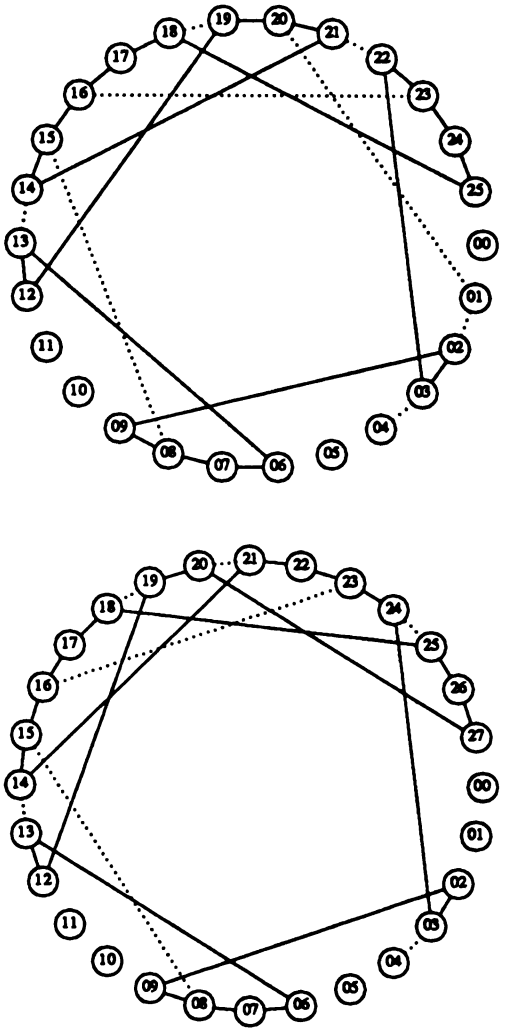

Figure 4-27. $F = \{0,5,10,11\}$.

Fault Set: 20 $F = \{0,5,12\}$. See Figure 4-28. Apply IB2 to the edge (13,14).

Fault Set: 21 $F = \{0,5,13\}$. See Figure 4-29. Apply IB6B between vertices 15 and 16.

Fault Set: 22 $F = \{0,5,14,15,16\}$. See Figure 4-30. Apply IB2 to the edge (17,18).

The previous fault-sets are sufficient (for this case) for $N=26$ and $N=28$. The following two complete this case for $N \geq 30$

Fault Set: 23 $F = \{0,5,2i-1,2i,2i+1\}$, where $14 \leq 2i \leq N-10$ and $N-2i \equiv 2 \, mod \, 4$. See Figure 4-31. Apply AB4.

Fault Set: 24 $F = \{0,5,2i-1,2i,2i+1\}$, where $14 \leq 2i \leq N-16$ and $N-2i \equiv 0 \, mod \, 4$. See Figure 4-32. Apply AB4.

This complete the case $\{0,5,z\}$.

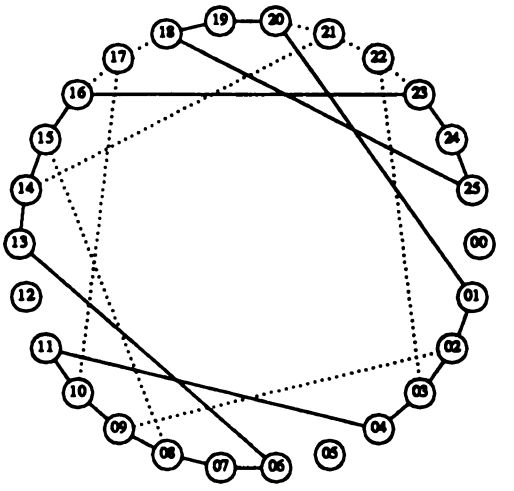

Figure 4-28. $F = \{0,5,12\}$.

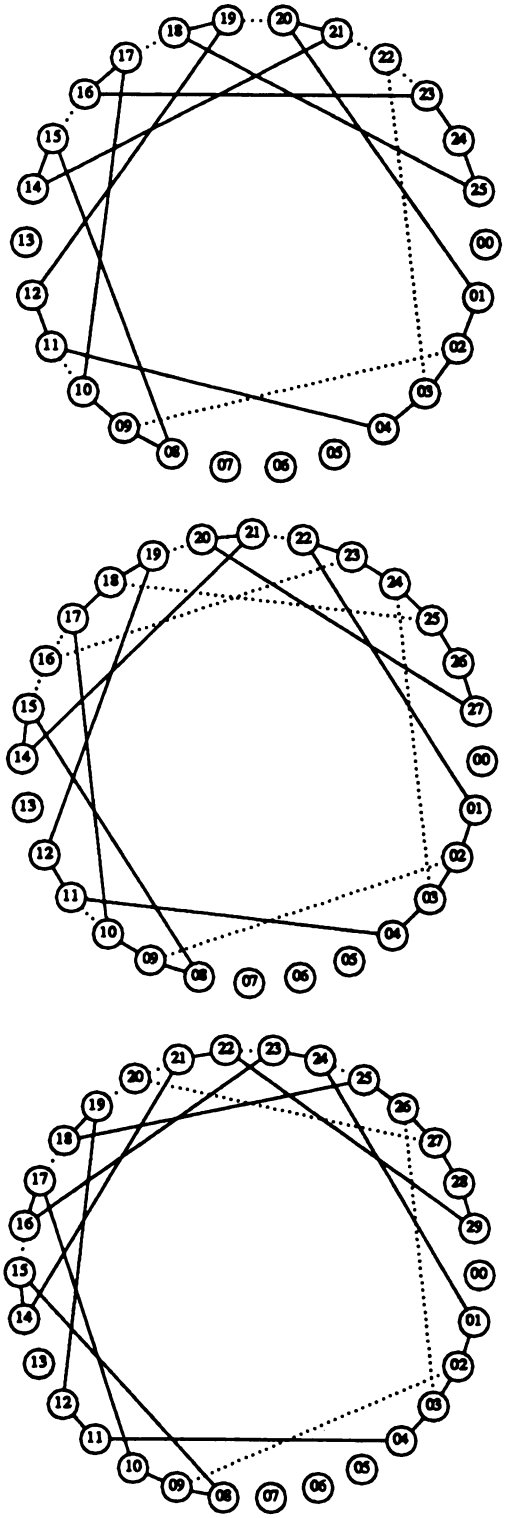

Figure 4-29. $F = \{0,5,13\}$.

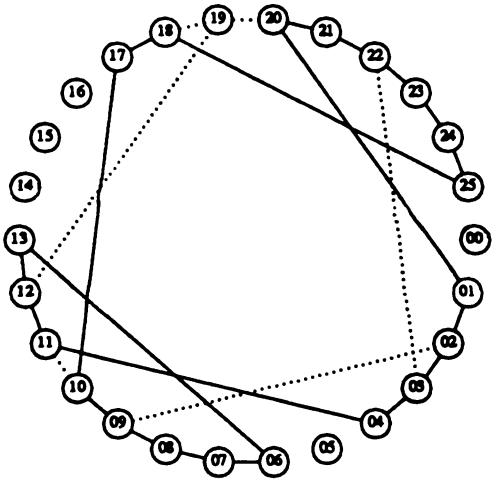

Figure 4-30. $F = \{0,5,14,15,16\}$.

CASE 5. $y \in \{6,7\}, 12 \leq z \leq N-6$.

Fault Set: 25 $F = \{0,6,7,12,13\}$. See Figure 4-33. Apply IB6B between $N-1$ and 0.

Fault Set: 26 $F = \{0,6,7,13,14\}$. See Figure 4-34. Apply IB6B between $N-1$ and 0.

Fault Set: 27 $F = \{0,6,7,8,15\}$. See Figure 4-35. Apply IB6B between 23 and 24.

Fault Set: 28 $F = \{0,6,7,N-7\}$. See Figure 4-36. Apply IB2 to the edge (17,18).

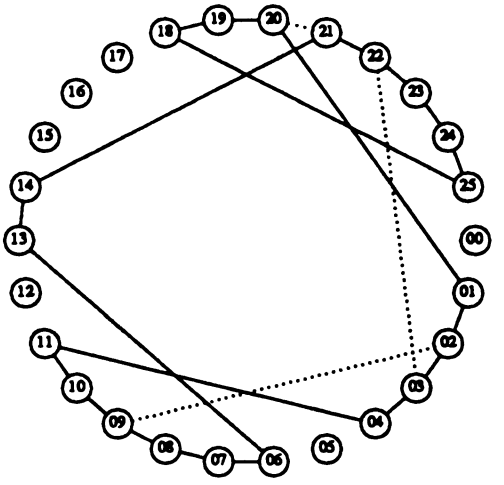

Figure 4-31. $F = \{0,5,2i-1,2i,2i+1\}, N-2i \equiv 2 \bmod 4$.

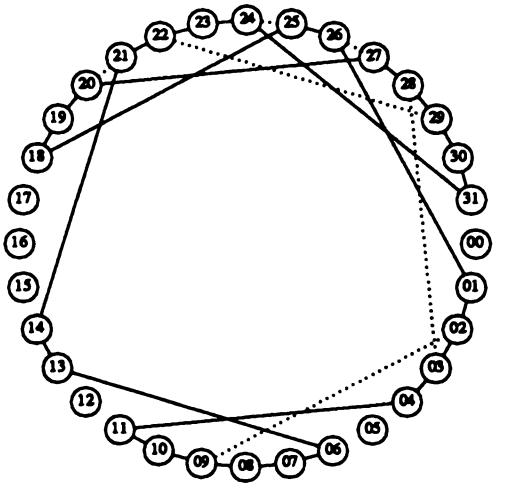

Figure 4-32. $F = \{0,5,2i-1,2i,2i+1\}, N-2i \equiv 0 \bmod 4$.

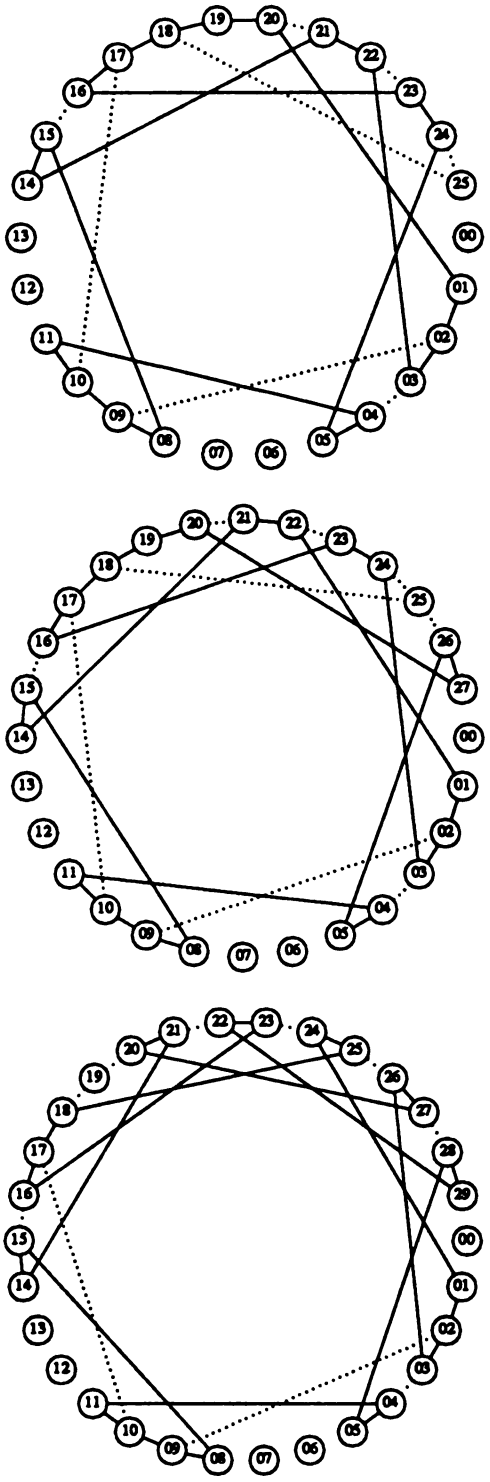

Figure 4-33. $F = \{0,6,7,12,13\}$.

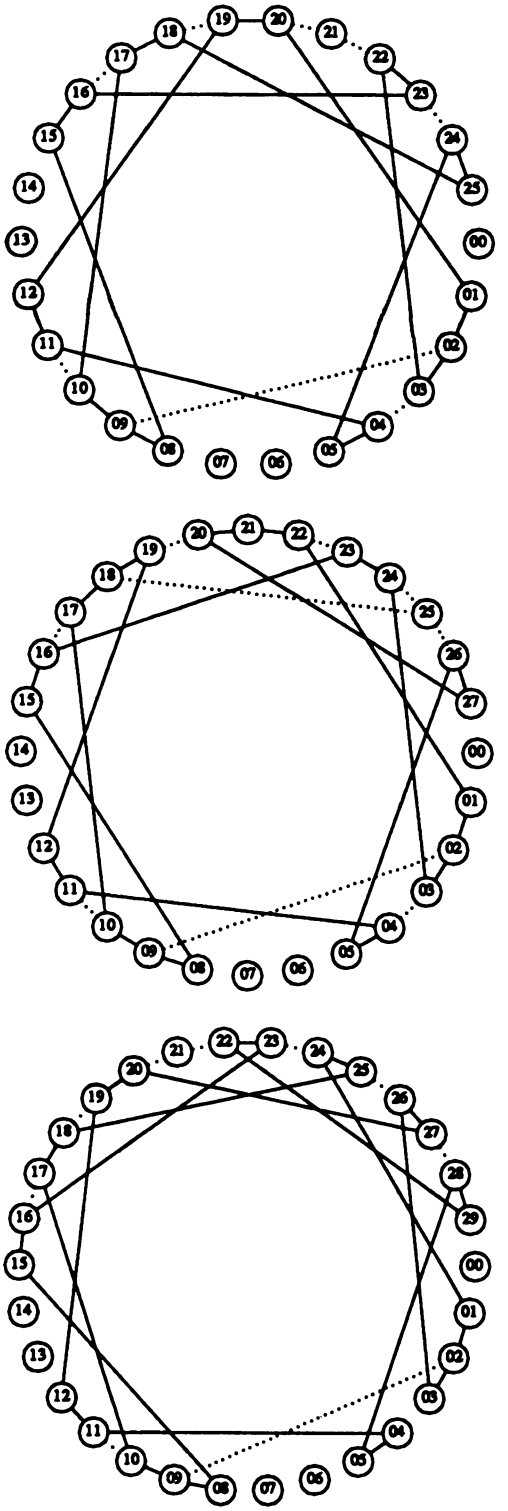

Figure 4-34. F = {0,6,7,13,14}.

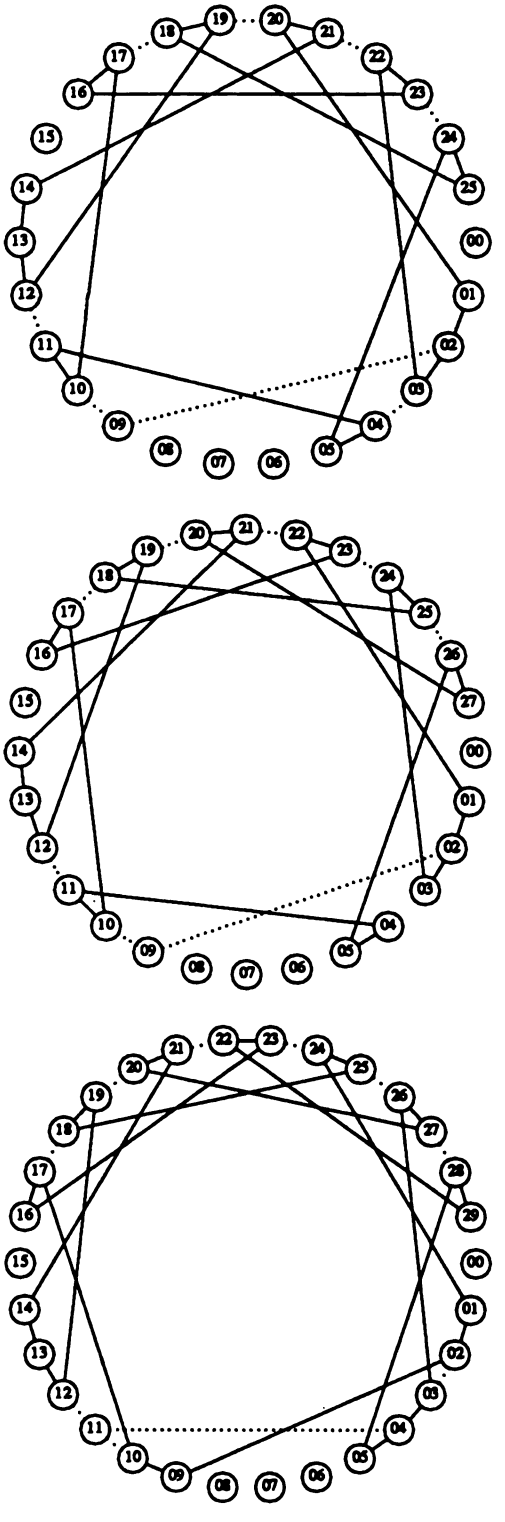

Figure 4-35. F = {0,6,7,8,15}.

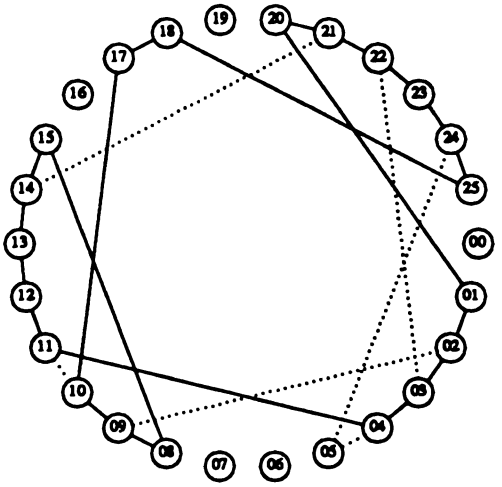

Figure 4-36. $F = \{0,6,7,N-7\}$.

Fault Set: 29 $F = \{0,6,7,N-8\}$. See Figure 4-37. Apply IB6B between vertices 17 and 18.

Fault Set: 30 $F = \{0,6,7,8,N-9\}$. See Figure 4-38. Apply IB2 to the edge (9,10).

Fault Set: 31 $F = \{0,6,7,4i,4i+1\}$. See Figure 4-39. The range of $4i$ is qualified as follows. $16 \leq 4i \leq N-12$, where $N-4i \equiv 0 \; mod \; 6$ (top), $16 \leq 4i \leq N-14$, where $N-4i \equiv 2 \; mod \; 6$ (middle), and $16 \leq 4i \leq N-16$, where $N-4i \equiv 4 \; mod \; 6$ (bottom). To see this result, first fix $4i = 16$. Then we may apply IB6B between vertices $N-1$ and 0. This verifies this choice of $4i$. Now note that we also apply IB4 between 11 and 12.

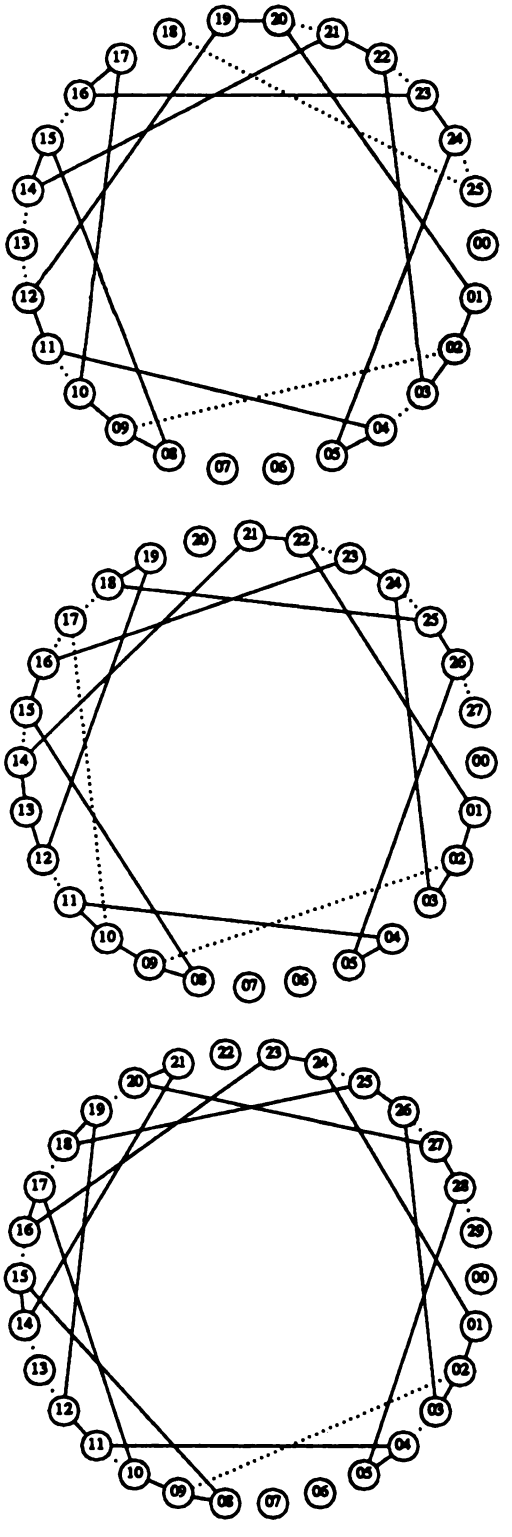

Figure 4-37. $F = \{0,6,7,N-8\}$.

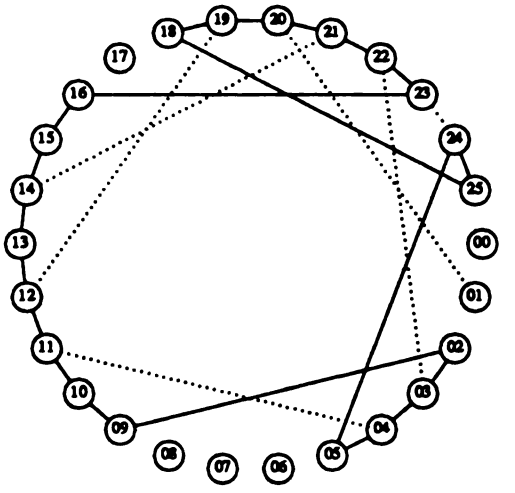

Figure 4-38. $F = \{0,6,7,N-9\}$.

Fault Set: 32 $F = \{0,6,7,4i+2,4i+3\}$. See Figure 4-40. The range of $4i$ is qualified as follows. $16 \le 4i \le N-12$, where $N-4i \equiv 0 \, mod 4$ (top), $16 \le 4i \le N-14$, where $N-4i \equiv 2 \, mod 4$ (bottom). To see this result, first fix $4i = 16$. Then we may apply IB4 between vertices 19 and 20. This verifies this choice of $4i$. Now note that we also apply IB4 between 7 and 8.

The above fault-sets cover all possible values of z except for $z = N-10$, which follows.

Fault Set: 33 $F = \{0,6,7,8,N-10\}$. See Figure 4-41. Apply IB4 between 7 and 8.

This completes the case $\{0,6,7,z\}$.

CASE 6. $y \in \{7,8\}, 16 \le z \le N-8$.

Fault Set: 34 $F = \{0,7,8,15,16\}$. See Figure 4-42. Apply IB6B between $N-1$ and 0.

Fault Set: 35 $F = \{0,7,8,N-9\}$. See Figure 4-43. Apply IB2 to the edge (13,14).

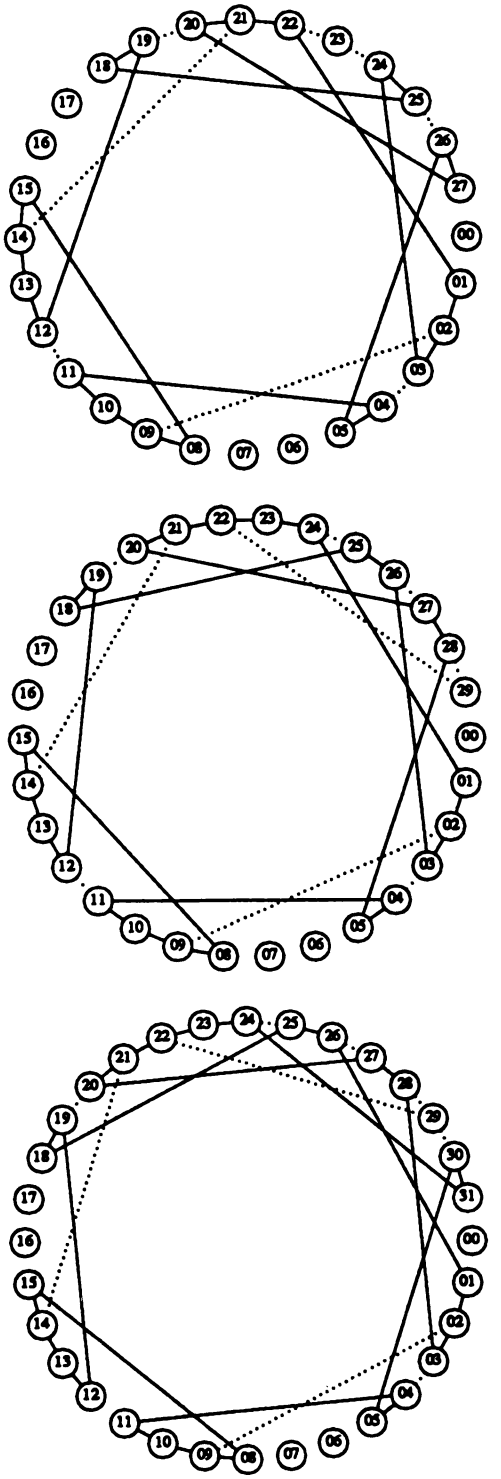

Figure 4-39. $F = \{0,6,7,4i,4i+1\}$.

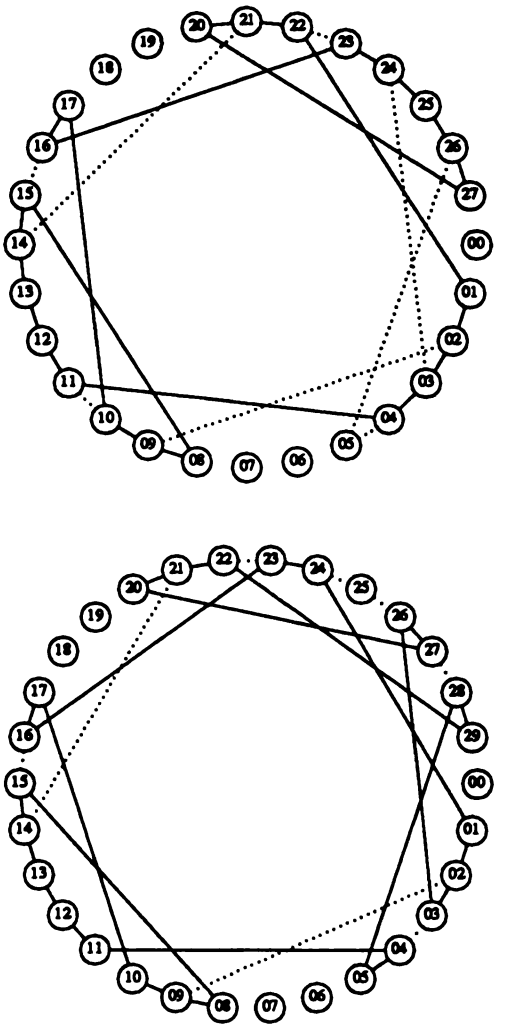

Figure 4-40. $F = \{0,6,7,4i+2,4i+3\}$.

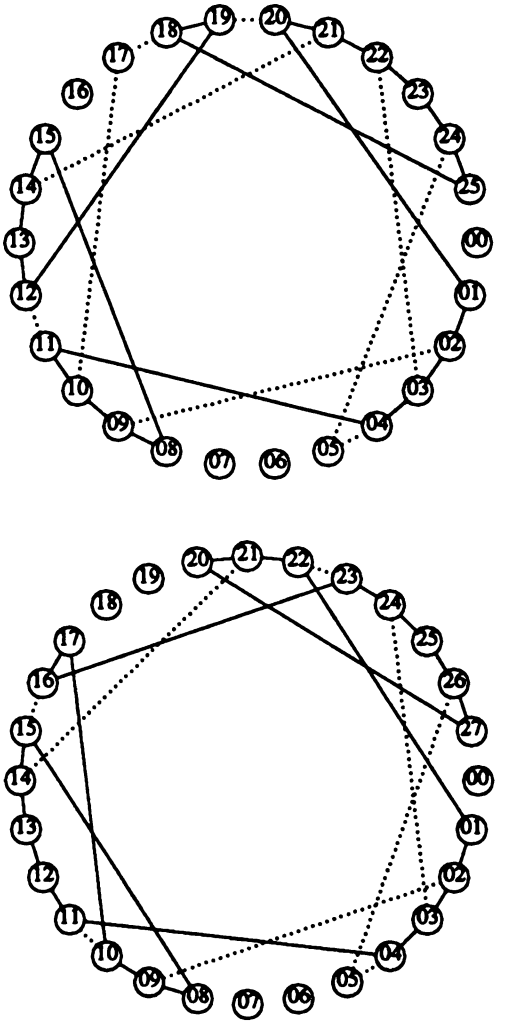

Figure 4-41. $F = \{0,6,7,N-10\}$.

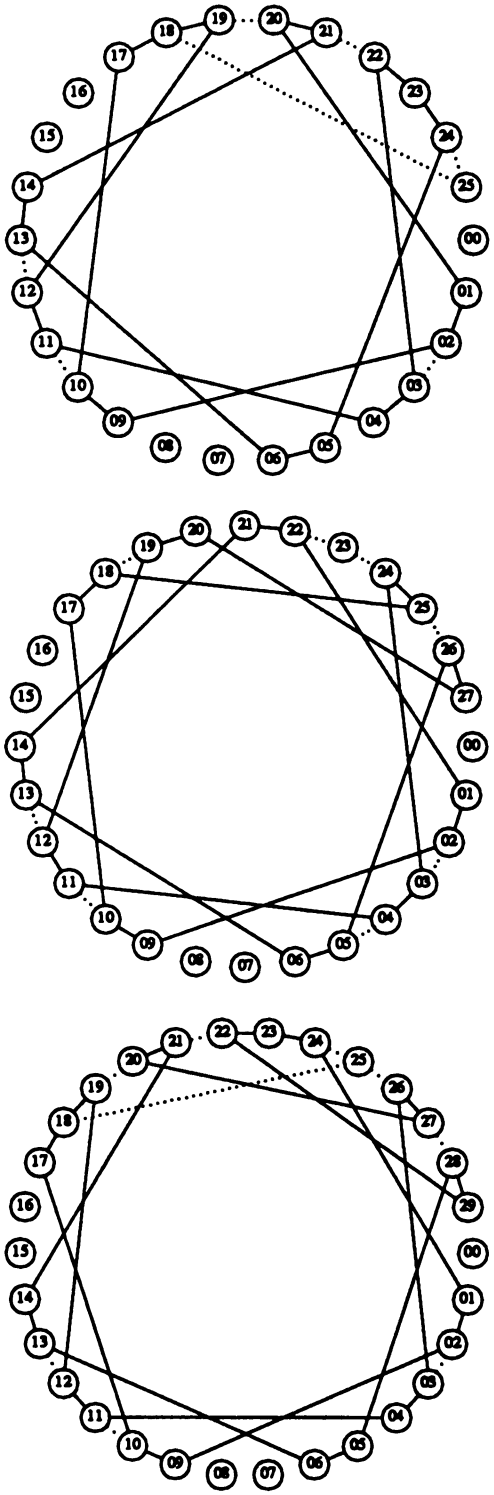

Figure 4-42. F = {0,7,8,15,16}.

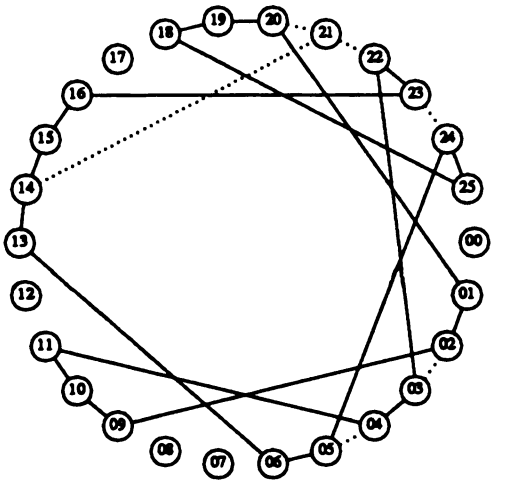

Figure 4-43. $F = \{0,7,8,N-9\}$.

We have thus far shown that any fault-set in which two vertices are at a distance of at most eight can be tolerated. It is not difficult to see that for $N = 26$, two faults must meet this requirement, and thus the proof is complete for

Fault Set: 36 $F = \{0,7,8,N-9,N-10\}$. See Figure 4-44. Apply IB4 to vertices 9 and 10.

For reasons similar to the above, the previous fault-set completes the proof for $N = 28$.

Fault Set: 37 $F = \{0,7,8,N-11\}$. See Figure 4-45. Apply IB2 to 9 and 10.

For reasons similar to the above, the previous fault-set completes the proof for $N = 30$.

Fault Set: 38 $F = \{0,7,8,2j+1\}$, where $17 \leq 2j+1 \leq N-11$. See Figure 4-46. We may apply IB2 to the edge (9,10) and also IB6B to the vertices $N-1$ and 0, allowing us to cover all odd vertices in the specified range.

Fault Set: 39 $F = \{0,7,8,20\}, N \geq 32$. See Figure 4-47. Apply IB4 to 25 and 26.

Fault Set: 40 $F = \{0,7,8,N-2j\}$, $10 \leq 2j \leq N-22$. See Figure 4-48. Apply IB2 to the edge (23,24).

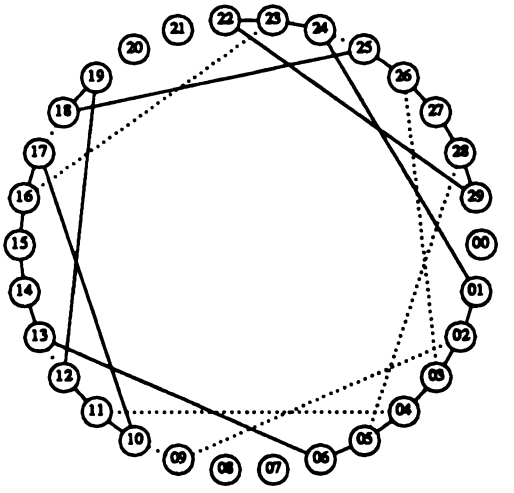

Figure 4-44. $F = \{0,7,8,N-9,N-10\}$.

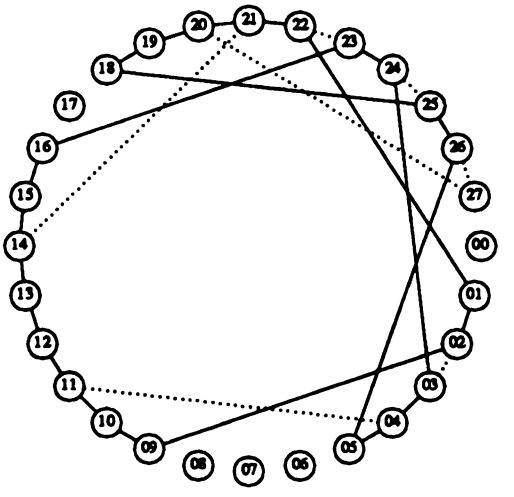

Figure 4-45. $F = \{0,7,8,N-11\}$.

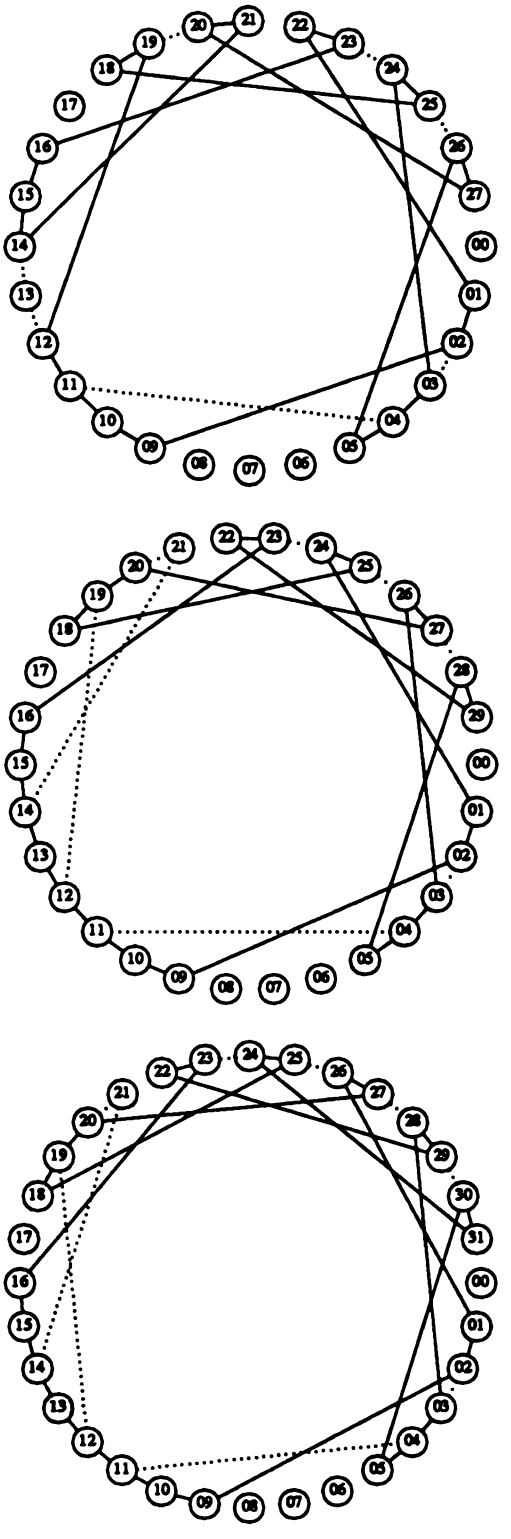

Figure 4-46. $F = \{0,7,8,2j+1\}$.

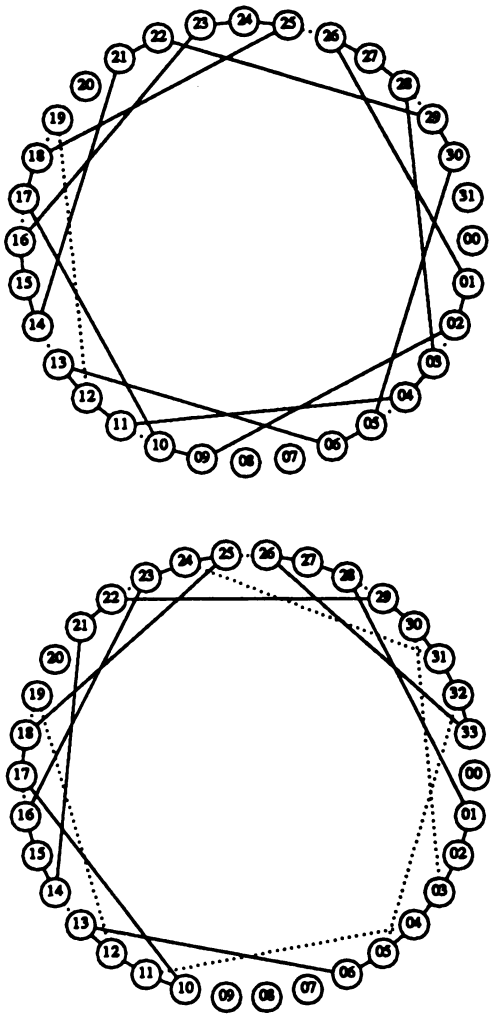

Figure 4-47. F = {0,7,8,20}.

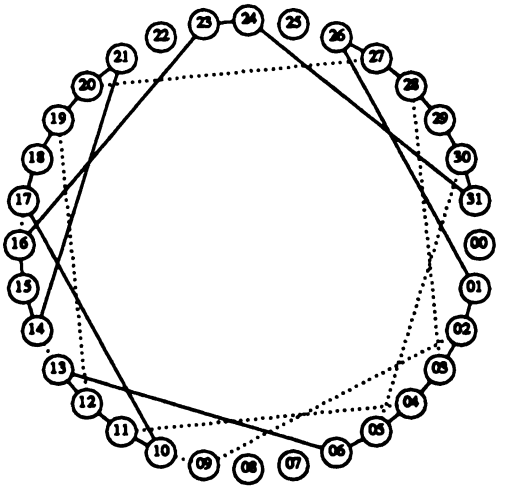

Figure 4-48. $F = \{0,7,8,N-2j\}$.

This completes the case of $\{0,7,8,z\}$ and we now consider our final two cases.

CASE 7. $y = 2i+1, 2i+1 \geq 9$

Fault Set: 41 $F = \{0,2i+1,2i+1+z\}$, where $9 \leq 2i+1 < N/2$ and $2i+1 \leq z \leq N/2+i$

See Figure 4-49. We may apply IB2 in the top graph to the edge (21,22); and also IB4 between vertices 17 and 18. In the bottom graph, apply IB2 to the edge (17,18) and IB4 between vertices 23 and 24. This covers all possible combinations of $\{0,2i+1,2j+1\}$ and $\{0,2i+1,2j\}$, for the necessary limits. We have thus proved that the failure of any combination of two even vertices and one odd vertex can be tolerated. By the symmetry of the graph, this also covers the case of two odd vertices and one even vertex, leaving only the case of three even vertices remaining, which is covered in case 8.

CASE 8. $y = 2j, 2j \geq 10$.

Fault Set: 42 $F = \{0,2j,2k\}$. See Figure 4-50. We may apply IB2 to the edges (1,2), (11,12), and (21,22) in any combination.

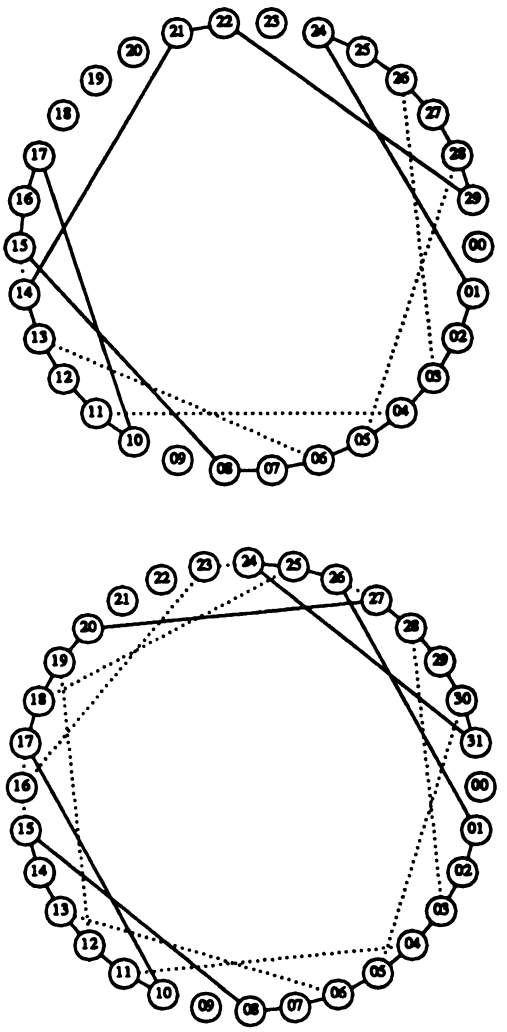

Figure 4-49. $F = \{0,2j+1,z\}$.

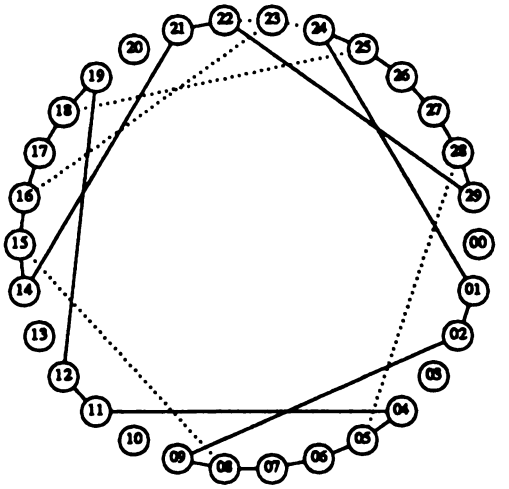

Figure 4-50. $F = \{0,2j,2k\}$.

We have now shown that every combination of three failures can be tolerated, finishing the proof. ■

4.2.3 Comparison with previous results.

In the previous section, we proved that cubic graphs, in particular chordal rings, can be used as designs for fault-tolerant loops and can tolerate up to 3 failures. Here, we compare our results with the work done in [Haye76]. For the 1-ft case, we have exhibited a class of 3-regular graphs which are in the class $\Gamma_1[C_N,2]$. Previously known results have Δ=4 and do not have the advantages of being regular. In the 2-ft case, our design is 3-regular, requires 4 spare vertices and $3(N+4)/2$ edges. Previously known results are 4-regular, use 2 spare vertices and require $2(N+2)$ edges. For the 3-ft case, our designs are 3-regular, require 6 spare vertices and $3(N+6)/2$ edges. Previous results are 5-regular, use 3 spares and $5(N+3)/2$ edges. Thus, the savings in edges is $O(N)$, while the cost in vertices is fixed (and small). So for large N, our designs yield improvements in terms of numbers of edges and are more in accord with the limitations of current

technology. In addition, since these designs have "extra" redundancy, they have the ability to tolerate certain specific patterns of greater than k failures. Indeed, in the proofs of the previous section, we demonstrated fault-sets of sizes $k+1$,..., $2k$ which could be tolerated.

It should also be noted that the design based on chordal rings is not unique; there are other families of cubic graphs which belong to the class $\Gamma_2[C_N,4]$, but which are not chordal rings. In particular, a certain subset of Generalized Petersen graphs is one such family [ZiEs89].

Chapter 5

Conclusion

5.1. Conclusions and Summary of Research Contributions

In this dissertation we have examined two problems relating to the topology of Multicomputer interconnection networks. The first of these is broadcasting; a communication paradigm in which one member of a network wishes to send a message to all the other members of the network. The second problem involves system-level fault-tolerance; a strategy for improving the reliability of a Multicomputer by designing it so that it can remain functional in the presence of faulty components (processors and communication links). Both of these problems were formulated in graph theoretic terms and extensive use of graph theory was made in establishing the results.

A software package for graph manipulation, developed at Michigan State University, was described in Chapter 2. This package allows users to easily create and manipulate graphs interactively. In addition, users may invoke algorithms on graphs to study their properties and test conjectures. This package was used in establishing some of the results in this dissertation and other recent research. It was also used in the preparation of figures for this dissertation and other recent publications.

It was noted that the topology of an MC plays a fundamental role in both the manner that broadcasting can take place, and how rapidly the process can be completed. The role of trees in the broadcasting process has been explored and we have provided a partial characterization of minimum broadcast trees. A recurrence relation has been developed which relates the maximum number of vertices in a graph which can be informed in a broadcast, to the maximum degree of the graph. This relation can be used in the characterization of both minimum broadcast trees and minimum broadcast graphs. The distribution of broadcast times over all trees of order up to 28 was given, showing that in "most" trees broadcasting can be done in the optimal or "near" the optimal amount of time. Finally, a distributed algorithm for broadcasting in Binary DeBruijn Graphs has been presented. The algorithm was shown to complete broadcasting in at most $(2log\,N)-1$ time steps, where N is the order of the underlying graph.

In the area of fault-tolerance, a new approach to system-level fault-tolerance for message-passing multicomputers has been presented. In this approach, a bound is placed on the number of connections allowed at each processor. It was shown that this approach necessitated the use of "extra" redundancy and designs based on chordal rings which can tolerate up to 3 processor failures were given. These designs illustrate that the new approach can be successfully applied to the design of fault-tolerant topologies and can achieve improvements over previous results.

5.2. Suggestions for Future Study.

There are a number of open problems of practical and theoretical interest in broadcasting in general networks, and in specific interconnection networks, e.g. BDN in particular. A partial list of problems which are immediate extensions of the results in Chapter 3 follows.

- Finding a tight upper bound for $bt(G)$

- Finding a relation between $bt(G)$ and other invariants of graph G.
- Completing the characterization of minimum broadcast trees.
- Analyzing and extending (beyond trees of order 28) the counting results for the broadcast time distributions given above.
- Generalizing the definition of local broadcasting to allow up to k messages to be issued concurrently from each informed node.

Many variations on DeBruijn graphs have been proposed [DuHw88]. One such variation allows the vertex set to be of any order (not just a power of two). In another variation, bases other than base 2 are used to define the interconnections. We conjecture that our broadcasting algorithm can be readily adapted for these and other variations.

In the area of fault-tolerance, we are currently investigating the applications of using *extra* redundancy in the development of general k-fault-tolerant designs for various topologies. The principal goal of this research is to achieve designs with feasible degree requirements. One basic area to consider is the relationship between the level of fault tolerance k, the maximum degree of the graph Δ, and the number of spare vertices.

Since we are using "extra" redundancy, it will be possible to provide, in a probabilistic sense, levels of fault tolerance which exceed the design specifications. It is an open question as to what probabilistic levels might be attainable.

The reconfiguration problem has not been addressed here. Although for any given set of failures a cycle can always be found, a general reconfiguration strategy is yet to be developed.

Finally, we have considered only vertex failures, however, it is clear from the proofs that the designs can also tolerate multiple edge failures. It would be valuable to extend the research to formally study edge failures and/or combinations of both.

BIBLIOGRAPHY

BIBLIOGRAPHY

[Agra83] D.P. Agrawal, "Graph theoretic analysis and design of multistage interconnection networks," *IEEE Trans. on Comput.*, Vol. C-32, pp. 637-648, 1983.

[ArLe81] B. Arden and H. Lee, "Analysis of Chordal Ring Network," *IEEE Trans. on Comput.*, Vol. C-30, pp. 291-295, April 81.

[BaKF86] Prithviraj Banerjee, Sy-Yen Kuo, W. Kent Fuchs, "Reconfigurable Cube-Connected Cycles," *Proc. of the Sixteenth Symposium on Fault-Tolerant Computing*, pp. 286-291, 1986.

[Barn68] G.H. Barnes et al., "The Illiac IV Computer," *IEEE Trans. on Comput.*, Vol. C-17, pp. 746-757, 1968.

[Batc80] K.E.Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. on Comput.*, Vol. C-29, pp. 336-340,1980.

[BeSi86] B. Becker and H.-U. Simon, "How Robust is the n-Cube," *Proc. of the 27th Annual Symposium on Foundation of Computer Science*, pp. 283-291, October 1986.

[BhAg83] L. Bhuyan and D.P. Argawal, "Design and Performance of generalized interconnection networks," *IEEE Trans. on Comput.*, Vol. C-32, pp. 1081-1090, 1983.

[Bien88] D. Bienstock, "Broadcasting With Random Faults," *Discrete Applied Math.*, Vol. 3 , pp. 4-7, 1988.

[Boes86] F.T. Boesch, "Graph Theory and Reliable Network Synthesis," *Technical Report*, Electrical Engineering and Computer Science Department, Stevens Institute of Technology, 1986.

[ChCD88] D.V. Chudnovsky, G.V. Chudnovsky, and M.M. Denneau, "Regular Graphs with Small Diameter as Models for Interconnection Networks," *Proc. of the 3rd International Conf. on Supercomputing*, Vol. III, pp. 232-239, 1988.

[Crow85] W. Crowther, et al., "Performance Measurements on a 128-node Butterly Parallel Processor," *Int'l Conf. Parallel Proc.," IEEE Computer Society Press, pp. 531-555, 1985.*

[Dall90] W.J. Dally, "Performance Analysis of k-ary n-Cube Interconnection Networks," *IEEE Trans. on Comput.*, Vol. 39, No. 6, pp 775-785, June 1990.

[DaSe87] W. Dally and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. on Comput.*, Vol. C-36, No 5. pp. 547-553, May 1987.

[DuHa88] S. Dutt and J.P. Hayes, "Design and Reconfiguration Strategies for Near-Optimal k-Fault-Tolerant Tree Architectures," *FTCS-18*, June 1988.

[DuHw88] D.Z. Du and F.K. Hwang, "Generalized de Bruijn Digraphs," *Networks*, Vol. 18, pp. 27-38, 1988.

[EsHa85] A.-H. Esfahanian and S.L. Hakimi, "Fault-Tolerant Routing in De Bruijn Communication Networks," *IEEE Trans. on Comput.*, Vol. 34, No. 9, pp. 777-789, September 1985.

[Esfa88] A.-H. Esfahanian, "Generalized Measures of Fault-Tolerance with Application to n-Cube Networks," *IEEE Trans. on Comput.*, Vol. 38, No. 11, pp. 1586-1590, Nov. 1989.

[EsNS89] A.-H. Esfahanian, L.M. Ni, and B. Sagan, "The twisted n-cube with Application to Multiprocessing," to appear in *IEEE Trans. on Comput.*, to appear.

[EsZi88] A.-H. Esfahanian and G. Zimmerman, "A New Fault Tolerance Analysis for N-cube Networks," *Proceedings of the International Symposium on Mini and Microcomputers*, December 1988.

[FaHe79] A.M. Farley, S.T. Hedetniemi, "Broadcasting in Grid Graphs," *Proc. 9th Conf. Combinatorics, Graph Theory, and Computing*, pp. 275-288, 1979.

[FaKr83] E.T. Fathi and M. Krieger, "Multiple Microprocessor Systems: What, Why, and When," *IEEE Comput.*, Vol. 16, pp. 23-35, March 1983.

[Farl79] A.M. Farley, "Minimal Broadcast Networks," *Networks*, Vol. 9, pp. 313-332 1979.

[Farl80] A.M. Farley, "Minimum-time Line Broadcast Networks," *Networks*, Vol. 10, pp. 59-70, 1980.

[Feng81] T. Feng, "A Survey of Interconnection Networks," *IEEE Comput.*, pp. 12-27, December, 1981.

[FHMP79] A. Farley, S Hedetniemi, S Mitchell, and A. Proskurowski, "Minimum Broadcast Graphs," *Discrete Math.*, Vol. 25, pp. 189-193, 1979.

[Flan77] P.M. Flanders et al., "Efficient High Speed Computing with the Distributed Array Processor," *High-Speed Computer and Algorithm Organization*, Kuch, Lawrie, and Sameh, eds., Academic Press, New York, 1977.

[Gott83] A. Gottlieb, "The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer," *IEEE Trans. on Comput.*, C-32, pp. 175-189, 1983.

[GrRe86] D.C. Grunwald and D.A. Reed, "Benchmarking Hypercube Hardware and Software," *Technical Report*, UIUCDCS-R-86-1303, Department of Computer Science, University of Illinois at Urbana-Champaign, 1986.

[GuHS86] J.L. Gustafson, S. Hawkinson and K. Scott, "The Architecture of a Homogeneous vector supercomputer," *Proc. of 1986 Int'l Conf. on Parallel Processing*, pp. 649-652, August 1986.

[Hara72] F. Harary, *Graph Theory*, Addison Wesley, 1972.

[Haye76] J.P. Hayes, "A Graph Model for Fault-Tolerant Computing Systems," *IEEE Trans. on Comput.*, Vol. 25, No. 9, pp. 875-884, September 1976.

[HMSC86] J.P. Hayes, T.N. Mudge, Q.F. Stout, S. Colley, S. and J. Palmer, "Architecture of a hypercube Supercomputer," *Proc. of 1986 Int'l Conf. on Parallel Processing*, pp. 653-660, August 1986.

[HeHL88] S. M. Hedetniemi, S. T. Hedetniemi and A. Liestman, "A survey of Gossiping and Broadcasting in Communication Networks," *Networks*, Vol. 18, pp. 319-349, 1988.

[Hill85] W.D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.

[HoJe81] R.W. Hockney and C.R. Jesshope, *Parallel Computers*, Adam Hilger, Ltd., 1981.

[HsYZ87] W.T. Hsu, P.C. Yew and C.Q. Zhu, "An Enhancement Scheme for Hypercube Interconnection Networks," *Proc. of the 1987 Int'l Conf. on Parallel Processing*, pp. 820-823, August 1987.

[HwBr84] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill Book Co., 1984.

[HwGh87] K. Hwang and J. Ghosh, "Hypernet: A Communication-Efficient Architecture for Constructing Massively Parallel Computers," *IEEE Trans. on Computers*, pp. 1450-1466, December 1987.

[JoHo89] S. Johnsson, Ching-Tien Ho, "Optimum Broadcasting and Personalized Communications in Hypercubes," *IEEE Trans. on Comput.*, Vol 38., No. 9, pp. 1249-1268, September 1989.

[KeKl79] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Comput. Networks*, Vol 3, pp. 267-286, 1979.

[KuRe80] J.G. Kuhl and S.M. Reddy, "Distributed Fault-Tolerance for Large Multiprocessor Systems," *Proc. 7th Annu. Symp. Comput. Architecture*, pp. 23-30, May 1980.

[KwTo81] C.-L. Kwan and S. Toida, "Optimal Fault-Tolerant Realizations of Some Classes of Hierarchical Tree Systems," *The 11th Int'l Conf. on Fault-Tolerant Computing*, pp. 176-178, 1981.

[Lars84] J. L. Larson, "An Introduction to Multitasking on the Cray X-MP-2 Multiprocessor," *Computer*, Vol. 16, No. 7, pp. 62-69, July 1984 Nov. 1988.

[LeHa88] T.C. Lee and J.P. Hayes, "Routing and Broadcasting in Faulty Hypercube Computers," *The 3rd Conf. on Hypercube Concurrent Computers and*

Applications, pp. 346-354, January 1988.

[LePe88] A. Leistman and J. Peters, "Broadcast Networks of Bounded Degree," *SIAM J. Disc. Math*, Vol. 1, No. 4, pp. 531-540, November 1988.

[Lies85] Arthur L. Liestman, "Fault-Tolerant Broadcast Graphs," *Networks*, Vol. 15, pp. 159-171, 1985.

[LoFu87] M.B. Lowrie and W.K. Fuchs, "Reconfigurable Tree Architectures using Subtree Oriented Fault Tolerance," *IEEE Trans. on Comput.*, Vol. 36, pp. 1172-1183, October 87.

[Pfis85] G.F. Pfister et al., "The IBM Research Parallel Processor Prototype (RP3)," *Proc. 14th Int'l Conf. on Parallel Processing*, 1985.

[PrRe82] D.K. Pradhan and S.M. Reddy, "A Fault-Tolerant Communication Architecture, for Distributed Systems," *IEEE Trans. on Comput.*, Vol. C-31, No. 9, pp. 863-869, September 1982.

[Prad81] D.K. Pradhan, "Interconnection Topologies for Fault-tolerant Parallel and Distributed Architectures," *Proc. 10th Int'l Conf. on Parallel Processing*, pp. 238-242, August 1981.

[Prad85] D.K. Pradhan, "Dynamically Restructurable Fault-Tolerant Processor Network Architectures," *IEEE Trans. on Comput.*, Vol. C-34, No. 5, pp. 434-447, May 1985.

[Prad86] D. Pradhan, Ed., *Fault Tolerant Computing, Theory and Techniques, Vol. I and II* Prentice Hall, 1986.

[PrVu81] F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computations," *Commun. ACM*, pp. 300-309, May 1981.

[PTLP85] J.C. Peterson, J.O. Tuazon, D. Lieberman and M. Pniel, "The Mark III Hypercube-Ensemble Concurrent Computer," *Proc. of the 1985 Int'l Conf. on Parallel Processing*, pp. 71-73, August 1985.

[Pros81] A. Proskurowski, "Minimum Broadcast trees," *IEEE Trans. on Comput.*, Vol C-30, No. 5, 1981.

[Quin87] M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill Book Company, 1987.

[RaAE84] C.S. Raghavendra, A. Avizienis, and M.D. Ercegovac, "Fault Tolerance in Binary Tree Architectures," *IEEE Trans. on Comput.*, Vol. 33, No. 6, pp. 568-571, June 1984.

[RaGA85] C.S. Raghavendra, M. Gerla, and A. Avizienis, "Reliable Loop Topologies for Large Local Computer Networks," *IEEE Trans. on Comput.*, Vol. C-34, No. 1, pp. 46-55, Jan. 1985.

[RCCT90] R.D. Rettberg, W.R. Crowther, P.T. Carvey and R.S. Tomlinson, "The Monarch Parallel Processor Hardware Design," *Computer*, Vol. 23, No. 4,

pp. 18-30, April 1990.

[SaPr89] M.R. Samatham, D.K Pradhan, "The De Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VSLI," *IEEE Trans. on Comput.*, Vol. 38, No. 4, pp. 567-581, Apr. 1989.

[SAFM88] C.L. Seitz, W.C. Athas, C.M. Flaig, A.J. Martin, J. Seizovic, C.S. Steele and W. Su, "The Architecture and Programming of the Ametek Series 2010 Multiprocessor," *Proc. of The 3-rd Conf. on Hypercube Concurrent Computers and Applications*, pp. 33-38, January 1988.

[ScWu84] Peter Scheuermann and Geoffrey Wu, "Heuristic Algorithms for Broadcasting in point-to-point Computer Networks," *IEEE Trans. on Comput.*, Vol C-33, No. 9, 1984.

[Seit85] C. Seitz, "The Cosmic Cube," *Commun. of ACM*, Vol. 28, No. 1, pp. 22- 33, January 1985.

[SlCH81] P.J. Slater, E.J. Cockayne, and S.T. Heditniemi, "Information dissemination in trees," *SIAM J. Comput.*, Vol 10, No. 4, pp. 692-701, 1981.

[Sten88] Per Stenstrom, "Reducing Contention in Shared-Memory Multiprocessors," *Computer*, Vol. 20, No. 11, pp. 26-37, Nov. 1988.

[Tane88] A.S. Tanenbaum, Computer Networks, Second Ed., Prentice-Hall, Englewood Cliffs, N.J., 1988

[Topk89] D. Topkis, "All-to-All Broadcast by Flooding in Communication Networks," *IEEE Trans. on Comput.*, Vol. 38, No 9, pp. 1330-1333, September 1989.

[WROM86] R. Wright, B. Richmond, A. Odlyzko, and B. McKay, "Constant Time Generation of Free Trees," *Siam J. Comput.*, Vol 15., No. 2, pp. 540-548, May 1986,

[YaHa86] R.M. Yanney and J.P. Hayes, "Distributed Recovery in Fault-Tolerant Multiprocessor Networks," *IEEE Trans. on Comput.*, Vol. 35, pp. 871-880, October 1986.

[ZiEs88] G. Zimmerman, and A.-H. Esfahanian, "GMP: A Graph Manipulation software Package for SUN Workstations," *Technical Report, MSU-ENGR-88-019*, Department of Computer Science, Michigan State university, October 1988.

[ZiEs89] G. Zimmerman and A.-H. Esfahanian, "A New Approach to System-Wide Redundancy in Designing Fault-Tolerant Topologies," *Technical Report, MSU-CPS-ACS-019* , Department of Computer Science, Michigan State University, February 1989.