INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

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

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

A

Linear Recursive Networks

By

Thami Smires

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2000

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Approval Page

This manuscript has been read and accepted for the graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

12/17/99
Date

Chair of examining Committee

Executive Officer

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Linear Recursive Networks

by

Thami Smires.

Advisor: Professor S.A. Ghozati

In recent years, the class of Linear Recursive Networks, LRNs, which first appeared in [26], has been studied by many researchers [16, 18, 20]. This thesis presents an extensive study of LRNs and their variances. It describes the static as well as dynamic properties of LRNs. In addition, the definition of the class of LRNs has been modified to cover a wider range of interconnection topologies. This has been accomplished by extending the alphabet over which the node labels of an LRN is generated to cover both numerals and non-numeral characters.

A hierarchy of LRNs has been defined and an important member of the class of LRNs, called the HLRN, has been studied exclusively. It is shown that the diameter of an n-dimensional HLRN is upper-bounded by $O(n)$.

Several efficient routing strategies are developed for the class of LRNs. Some routing algorithms are applicable to all LRNs. Others only apply to some members of the class of LRNs.

Embedding properties of LRNs have been studied. Algorithms to embed an m-dimensional LRN into an n-dimensional one are presented. . In particular, the embedding of an n-cube into an n-LRN is described. Partitioning of LRNs, which is essential in multiprocessing environment, is discussed. It is shown that the partitioning is process, which is controlled by the generator of the LRN, A.

Furthermore, a new cube-like interconnection topology, *n-fastcube*, is developed, The topological and routing properties of fast-cubes are investigated. The diameter of the fastcube if almost half of the diameter of the cube while its cost is the same as the cost of the cube. In the same context, a methodology to reduce the diameter of interconnection networks is introduced. Finally, fault- tolerance of LRNs is discussed and ways to handle node failures are presented.

Table of Contents

Lists of Figures

Chapter 1

Since the introduction of the first generation of computers in early 1945, computer technology has improved considerably. Early computers were bulky and expensive and only a few large organizations could afford them. And for the most part, these computers operated independently.

Starting in the mid-1980, the invention of high-speed computer networks has revolutionized the computer industry. It is now possible to interconnect several thousands of computers/processors to form a parallel computing system. In what follows, we will refer to computers/processors as Processing Elements or simply PEs.

The performance of a computing system has always been an important design issue. In recent years, with the possibility of connecting several computers by a LAN (Local Area Networks), or integrating several processors on the same board, the performance of the resulting system has improved significantly. In addition to the performance enhancement, a multiprocessing system is inherently fault tolerant, i.e., in the case of one or more processing element's failure the system remains functional. The number of faulty processing element's which are tolerated varies and depends on the architecture of the system.

In general a parallel computer provides a more cost-effective solution than a sequential uniprocessor of comparable power. This fact, however, was not universally recognized in the early years of Parallel Processing. Not long ago, computer scientists believed that

it was more cost-effective to buy a more expensive CPU. This fact known as Grosch's law [29] can be stated as follows:

The computing power of a uniprocessor is proportional to the square of its price. It turns out that Grosch's law is not applicable to computing systems with several interconnected processors. It is much more cost/performance-efficient to put together several non-expensive CPUs than having an expensive single CPU.

In a multiprocessing system, the processors need to synchronize their operations. Therefore interprocessor communication networks are needed to provide necessary communications between processing elements. Interconnection networks are also essential in load sharing operations. Load sharing in a parallel computing system permits fair and uniform distribution of tasks between processor.

In recent years, a large number of routing protocols have been developed for interprocessor communication. The design of these protocols is greatly influenced by the underlying interconnection topologies through which this communication occurs. Messages/packets of information are exchanged between these processors via interconnection networks. Several interconnection topologies have been extensively studied and their topological as well as routing properties have been reported in the literature [24, 7, 12]. A comprehensive review of some of the most common interconnection topologies will be presented in the next chapter.

1 Preliminaries

1.1 Birth of interconnection networks.

A variety of tasks require computational power and speed up which is made possible by parallel processing. These tasks include Weather forecasting, map making, missile guidance, ...

Parallel systems are comprised of a number of tightly coupled cooperating processors that can supply the necessary computing power. The age of very Large-scale integration (VLSI) has made it possible and feasible for large-scale parallel/distributed systems. For example, the MMP (massively Parallel Processor) [25] consists of 2^{14} processors.

The major problem in designing large-scale parallel/distributed systems is the interconnection network that will provide the interprocessor communication and the memory access for the processors. The interconnection scheme has to provide fast, flexible and reliable communications at reasonable costs. Obviously the traditional single shared bus system (Figure 1) is not sufficient since we desire that processors exchange data simultaneously.

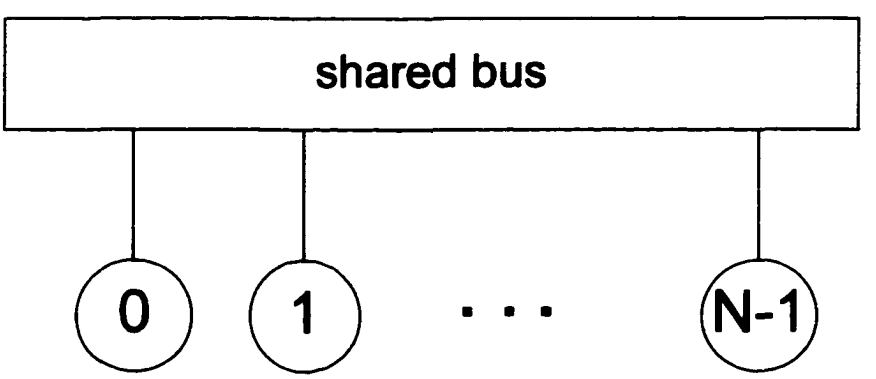


Figure 1 : A single shared bus, providing communications for N devices..

So we need an environment in which processors could communicate simultaneously. An extreme case is a completely connected system. Two possible configurations are possible. In the first configuration, the processing elements are connected among themselves. In the second one, the processing elements are connected with memory modules.

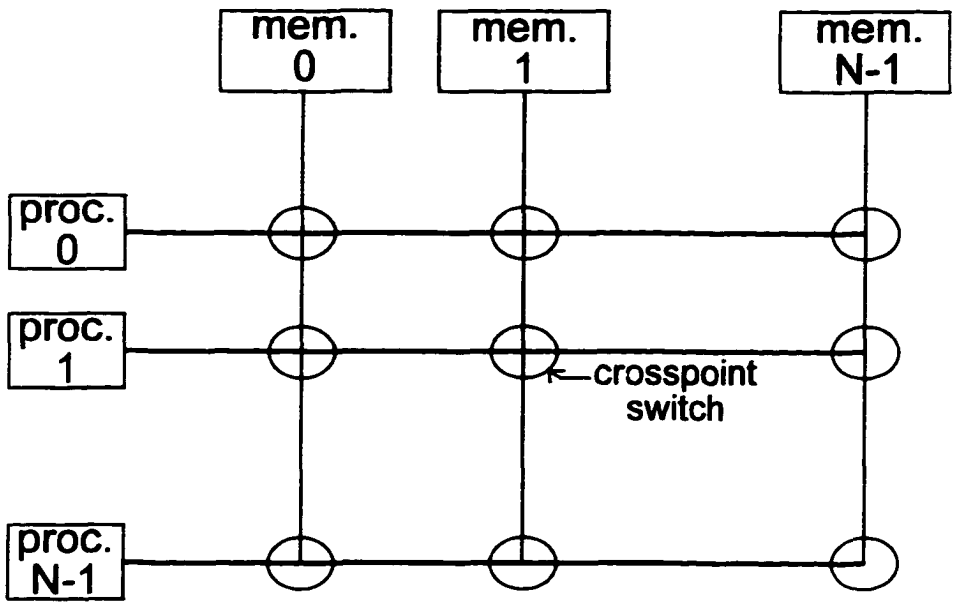


Figure 2: A crossbar Network connecting N processors to N memories.

Figure 2 illustrates the first configuration while Figure 3 illustrates the second. In either configuration costs are huge. In the first the need of (N-1) lines for each processor which adds up to N(N-1) lines.

In the second, the crossbar, N^2 crosspoint switches are needed. These interconnection schemes provide very good communication but at a very high cost. Interconnection schemes that provide fast reliable communication at a relatively cheap cost are desired. Several interconnection topologies are presented in the literature.

We will include later a review of the common interconnection Networks.

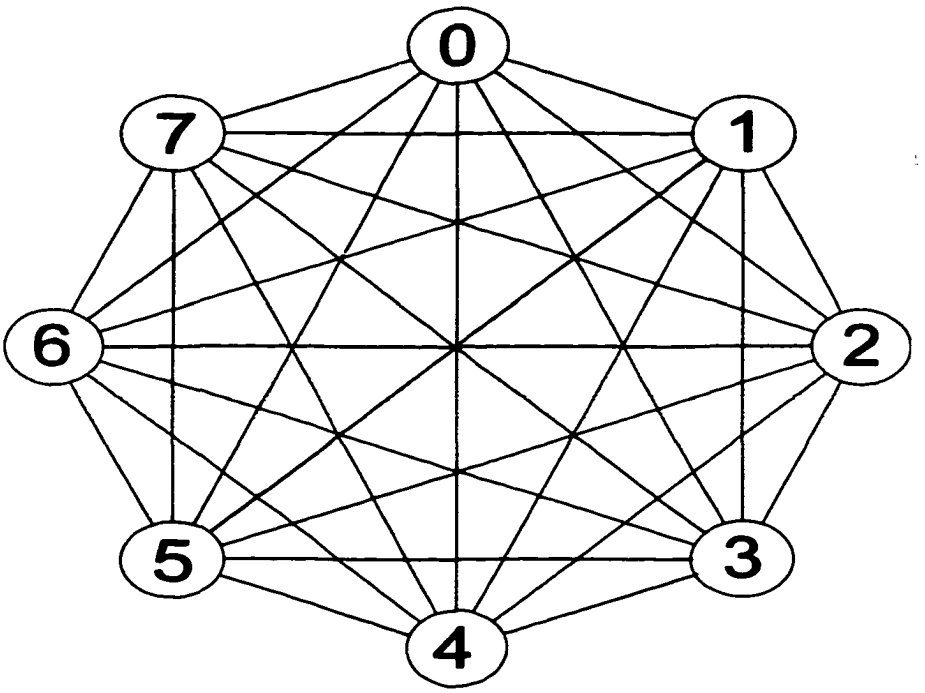


Figure 3: A completely connected system.

1.2 Attributes of Interconnection Networks.

In this section we will describe some of the more important, properties of interconnection networks.

Usually, the topology of an interconnection network is represented by a graph G = (V, E), where V is the set of vertices i.e. nodes of the interconnection network and E is the set of edges i.e. links in the interconnection network. The nodes denote the processing elements and the links are the physical links that interconnect the Processing elements.

Maximal degree: Degree of a node PE_i, denoted by degree(PE_i), is the number of Processing elements directly connected to it in the interconnection network. The maximal

degree of an interconnection network is $\max_{0<i<N}(\text{degree}(PE_i))$ where N is the number of Processing elements. To reduce the implementation cost of a network, it is a desirable property to have a small maximal degree.

Diameter: Diameter of a network is defined as the longest of the shortest paths between any two Processing elements in an interconnection network.

Let *path* be a 2-d matrix that represents the shortest path between any two Processing elements PE_i and PE_j, then the diameter of an interconnection network is $\max_{0<i<N, 0<j<N}(\text{path}(PE_i, PE_j))$.

Throughput: Throughput denotes the maximum number of processing elements that can send data simultaneously during the transfer step.

Symmetry: An interconnection network is said to be symmetric if it "looks the same" from every node. With symmetric networks we can use identical processors since every node plays an identical role in the system. This property simplifies the design of algorithms because any node can be the starting node. Any routing function that works for a certain processing element can be extended to all other Processing elements. Symmetry is also useful in partitioning interconnection network. Another aspect in which symmetry is useful is congestion during message routing. The edge symmetry ensures that communication load is uniformly distributed so there is no congestion at any one point.

Single-Stage and Multi-Stage: There are two main types of interconnection networks based on the number of intermediate links between processing elements. If there is a

direct connection between at least two Processing elements we refer to that network as a single-stage network. If between any two Processing elements we have at least one or more stages of switches, we talk about a multi-stage network. I.e. interconnection networks can be constructed a single stage of switches or a multiple stages of switches [24, 25]. In a single-stage network, data items may have to be passed through the switches several times before reaching their final destination. In a multi-stage network, one pass through the multiple stages of switches is sufficient to transfer the data items into their final destinations. Many of the documented interconnection networks have both a single-stage and a multi-stage implementation. The advantage of multi-stage networks over single-stage networks is the constant delay, i.e. the time it takes the message to go from point A to point B. Since in a multi-stage implementation the number of stages is fixed, therefore the delay is a constant. In a single-stage interconnection network, the delay is not fixed and late responses may be interpreted in a variety of ways.

Scalability: Scalability reflects the ability of a network to grow, i.e. to be able to add more Processing elements to the network while retaining the same topology. There are several problems related to this property. How to connect the new nodes at a minimum cost, and what is the minimum number of Processing elements that has to be added. Scalability is one of the down sides of the Hypercube topology to be explored later.

Fault Tolerance and Reliability: Processing elements can stop functioning for a multitude of reasons either defects or power failure... In case of such a failure, a mechanism should to cope with this eventuality. Reliability is the measure of the network to tolerate such failures. There are two approaches: hardware and software. A hardware solution dictates to introduce redundancy in the system so that in case of failure of one,

control is passed to the other and therefore the system can survive to the failure. A software solution will dynamically reconfigure the system so that it will emulate the original network with some decrease of the system's performance.

Important Interconnection Networks.

In this section, we will describe some of the most reported interconnection topologies in the literature.

Ring: Ring topology is one of the first interconnection networks to be implemented. It is a simplified version of the completely connected system where every node is connected

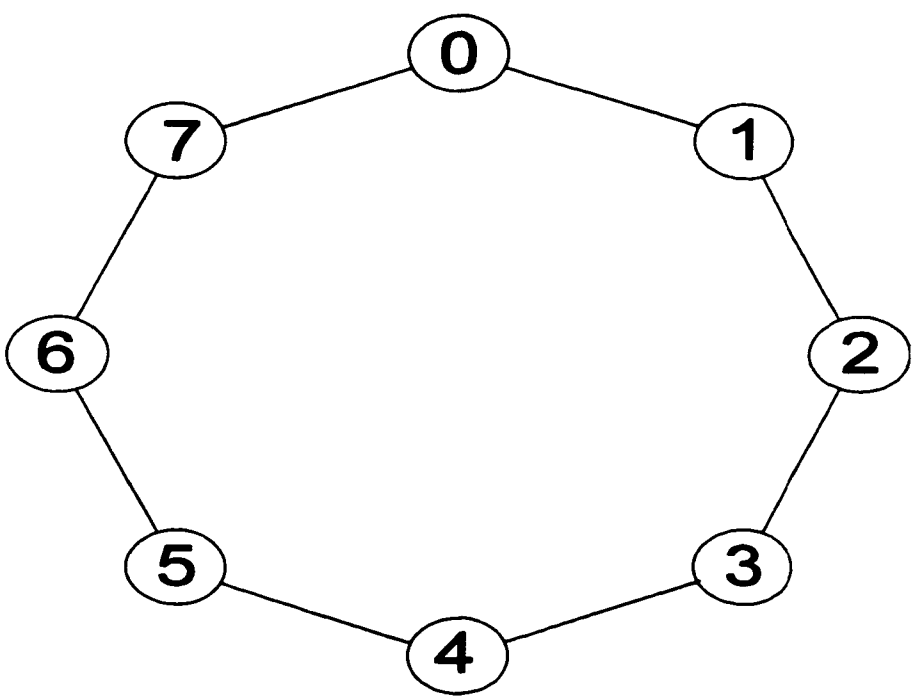


Figure 4: Ring topology with N= 8.

only to its two neighbors. A ring topology consists of a set of nodes connected by point to point links in a closed loop. These nodes serve as attachment points in a ring topology as they attach computers to the ring. They are also called repeaters because after these nodes receive a message, they retransmit that same message to the next node until the node, which the message was originally addressed to receives the message. Once the destination node successfully reads the message, it sends message back around the ring to the sender acknowledging the message. Figure 4 shows a ring network with N = 8.

Illiac: The Illiac network of N nodes consists of four interconnection functions defined as follows:

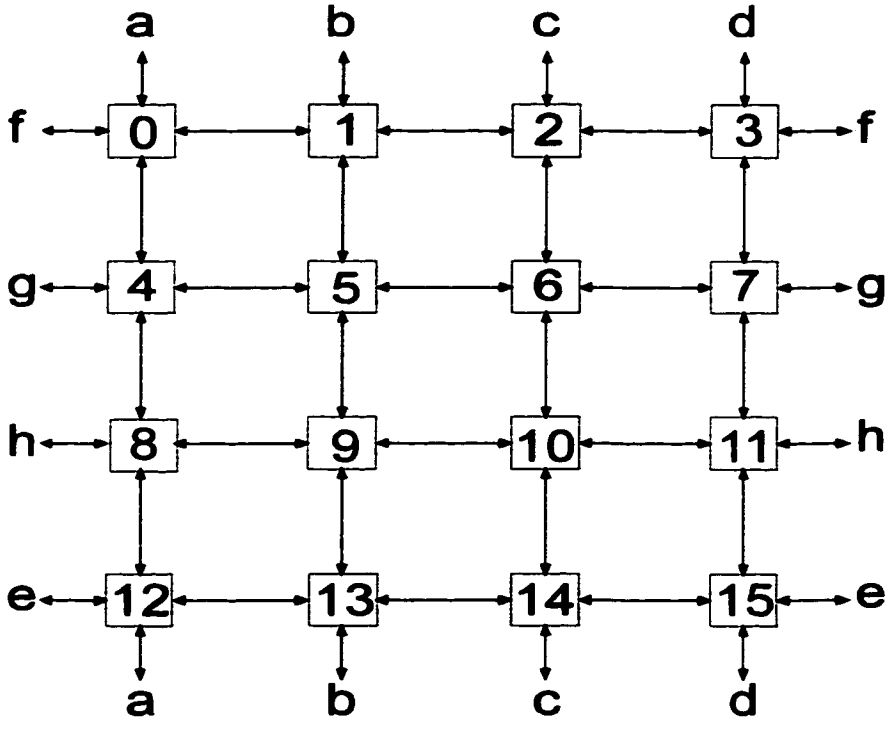


Figure 5: Illiac Network for N = 16.

$Illiac_{+1}(P) = (P + 1) \text{ Mod } N$

$Illiac_{-1}(P) = (P - 1) \text{ Mod } N$

$Illiac_{+n}(P) = (P + n) \text{ Mod } N$

$Illiac_{-n}(P) = (P - n) \text{ Mod } N$

where n = sqrt(N) and is assumed to be an integer. If the Processing elements are considered to be an n x n array, then each processing element is connected to its four neighbors as in Figure 5.

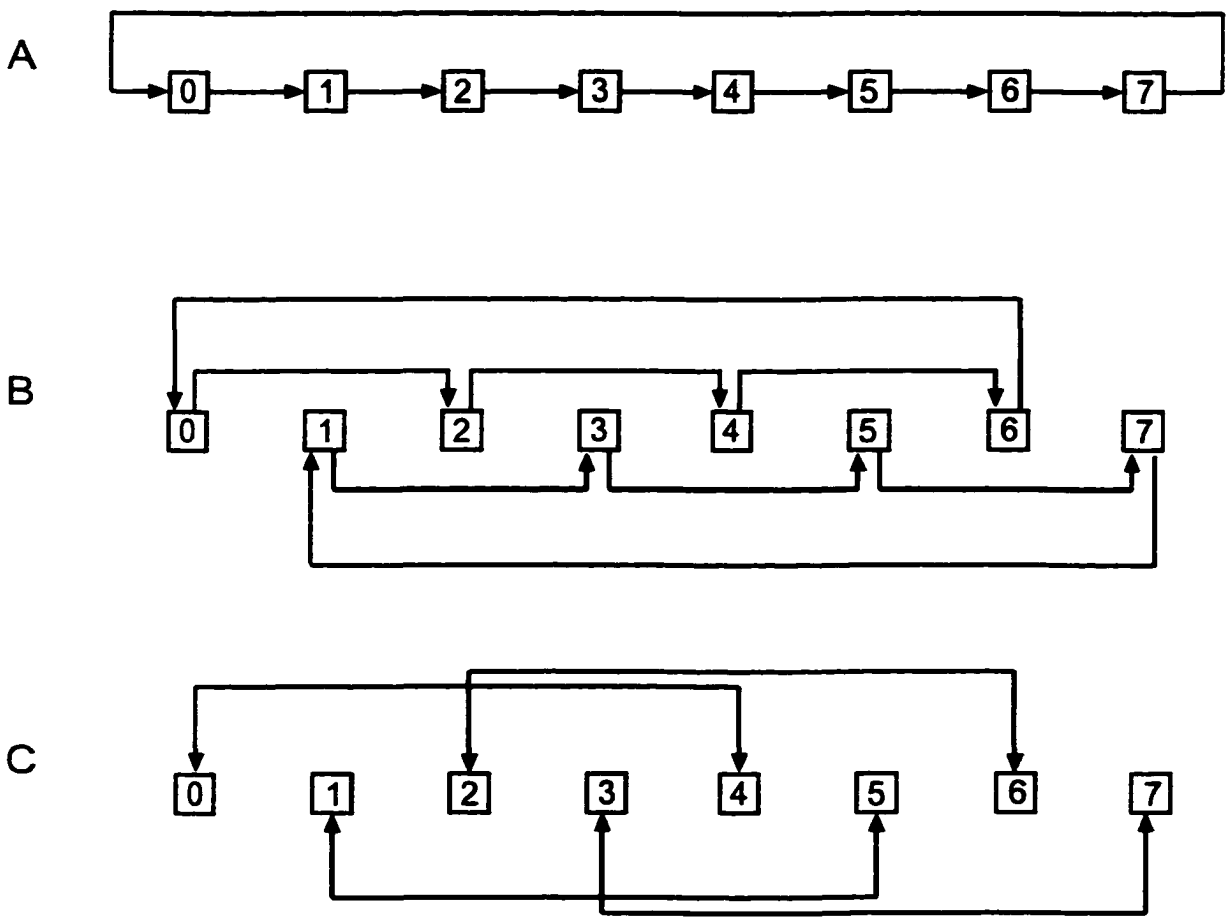


Figure 6: PM2I Network for N = 8. A. PM2 +0. B PM2 +1. C. PM2 +2.

PM21: The Plus-Minus 2^i (PM2I) network of N Processing elements consists of 2m interconnection functions defined by:

$PM2_{+i}(P) = P + 2^i \bmod N$

$PM2_{-i}(P) = P - 2^i \bmod N$

for $0 \leq i < m$. Figure 6 shows the PM2I interconnections for N = 8.

Hypercube: An r-dimensional Hypercube has $N=2^r$ nodes and $r2^{r-1}$ edges. Each node corresponds to an r-bit binary string, and two nodes are connected if and only if they differ in a single bit. As a consequence, each node is incident to r = logN other nodes. The diameter of the cube is r. Figure 7 shows a Hypercube with N = 8.

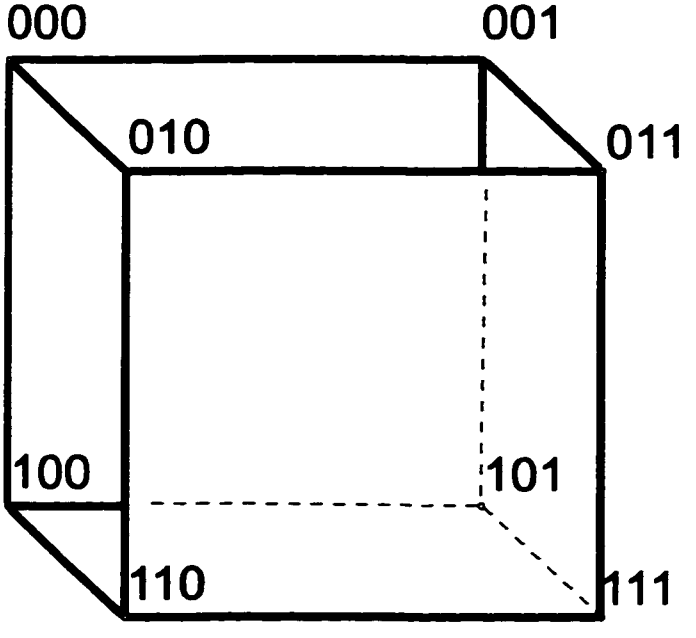


Figure 7: Three-dimensional Hypercube.

The Cube network consists of routing functions:

$Cube_i(a_{r-1}\ a_{r-2} \ldots a_{i-1}, a_i, a_{i+1} \ldots a_1\ a_0) = a_{r-1}\ a_{r-2} \ldots a_{i-1}\ a'_i\ a_{i+1} \ldots a_1\ a_0$ where $0 \leq i < r$.

The Hypercube has both a single-stage implementation (Figure 8), and a multi-stage implementation (Figure 9).

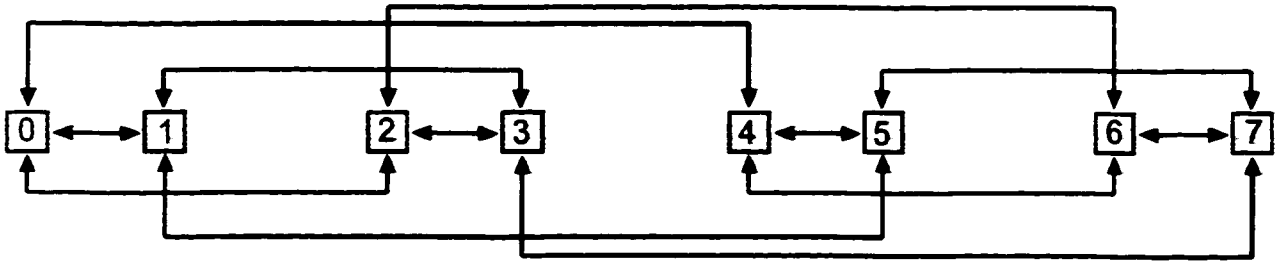


Figure 8: Single-stage implementation of a three dimensional Hypercube.

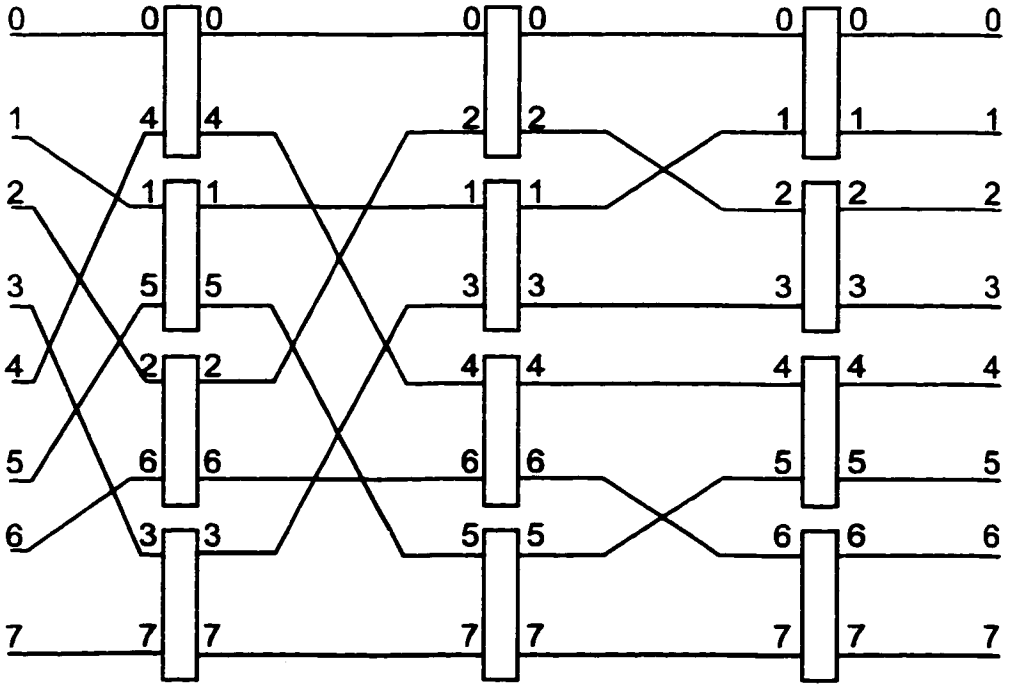


Figure 9: Multi-stage implementation of a three dimensional Hypercube.

Shuffle-Exchange: An r-dimensional shuffle-exchange graph has $N=2^r$ and $3*2^{r-1}$ edges. Each node corresponds to a unique r-bit binary number, and two nodes, u and v, are linked by an edge if either:

u and v differ precisely in the last bit

u is left or right cyclic shift of v.

If u and v differ in the last bit, the edge is called an exchange edge. Otherwise, the edge is called a shuffle edge. Figure 10 represents an 8-node shuffle exchange graph.

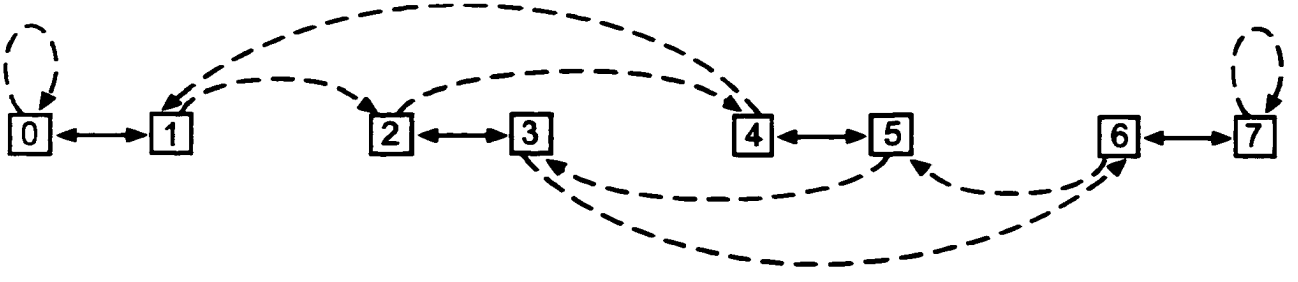


Figure 10: Shuffle-Exchange Network for N = 8.

The shuffle-exchange network consists of shuffle function and an exchange function:

Shuffle($a_{r-1}\ a_{r-2} \ldots a_1\ a_0$) = $a_{r-2}\ a_{r-3} \ldots a_1\ a_0\ a_{r-1}$.

Exchange($a_{r-1}\ a_{r-2} \ldots a_1\ a_0$) = $a_{r-1}\ a_{r-2} \ldots a_1\ a'_0$

Its multi-stage version called the shuffle or omega network is shown in Figure 11.

Butterfly: An r -dimensional butterfly has $(r + 1)2^r$ nodes and $r2^{r+1}$ edges. The nodes correspond to pairs (w, i) where I is the level or dimension of the node $(0 \leq i \leq r)$ and

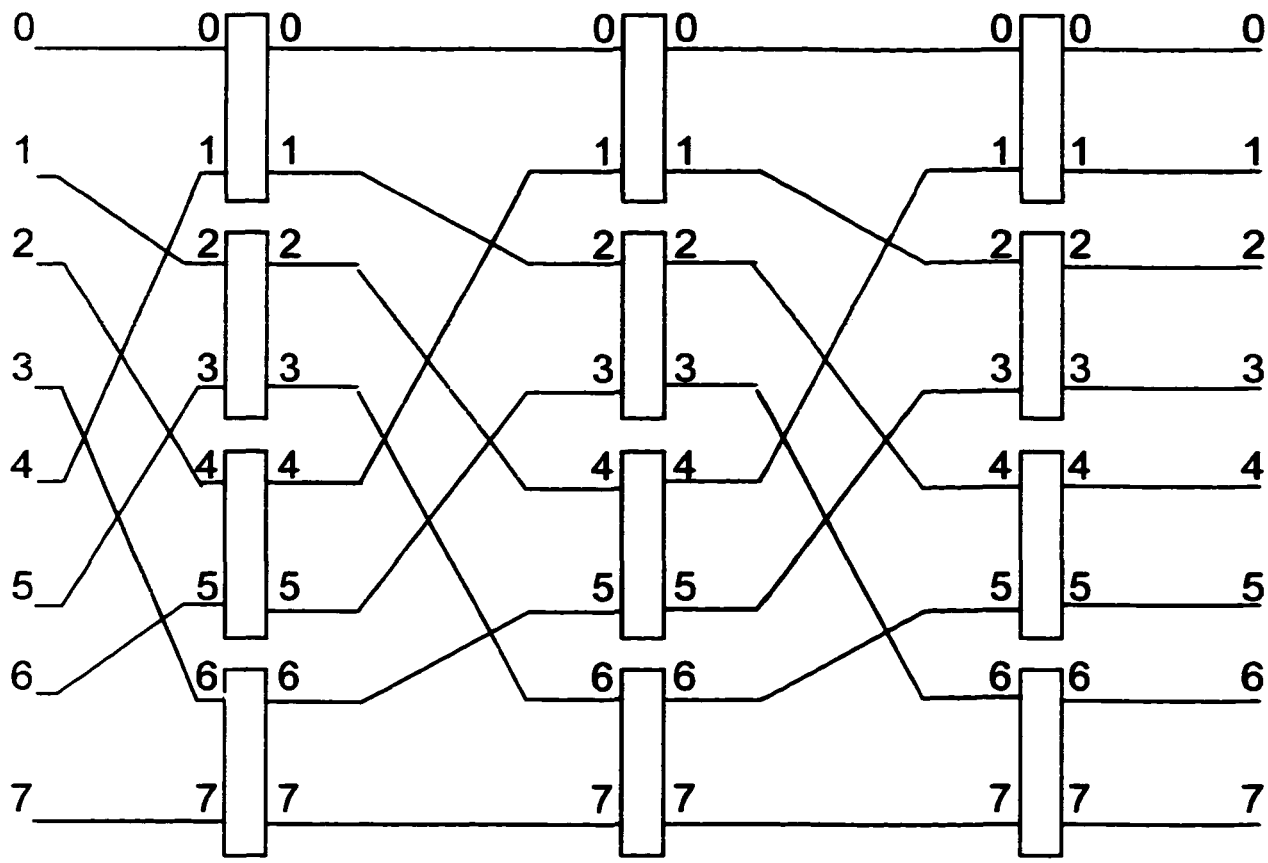


Figure 11: The Omega Network with N = 8.

w is an r-bit binary number that denotes the row of the node. Two nodes (w, i) and (w', i') are connected by an edge if and only if i' = i + 1 and either w and w' are identical, or w and w' differ in precisely the i'th bit. Figure 12 shows a three-dimensional butterfly.

2 Flynn's Taxonomy

There is no completely satisfactory characterization of the different types of parallel systems. Flynn defined the most popular taxonomy in 1966. The classification is based on the notion of a stream of information. Two types of information flow into a

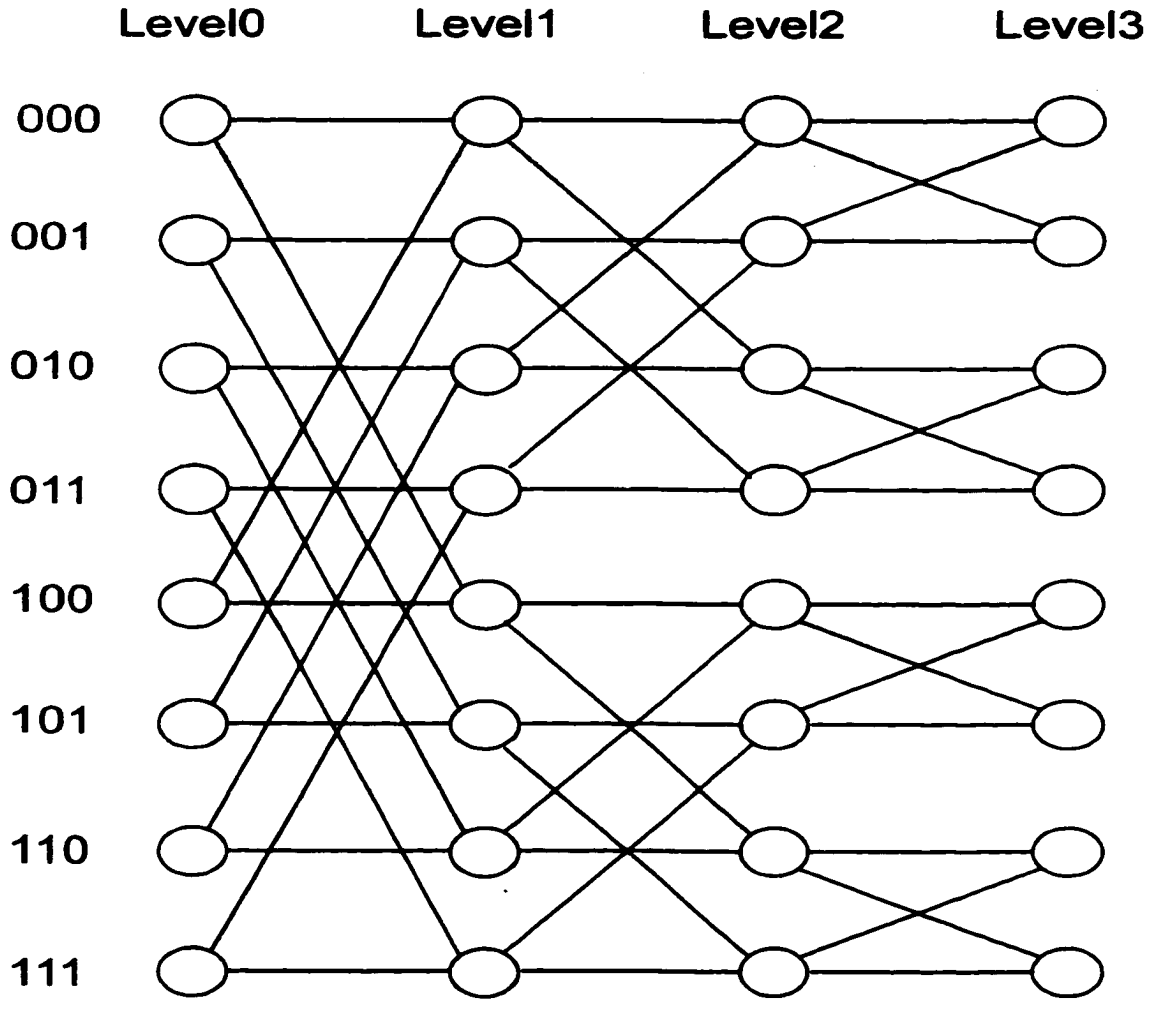


Figure 12: A three-dimensional butterfly.

processor: instructions and data. Conceptually these can be separated into two independent streams, whether or not the information actually arrives on a different set of wires. Flynn's taxonomy classifies machines according to whether they have one stream

or more than one stream of each type. The four combinations are SISD (single instruction stream, single data stream), SIMD (single instruction stream, multiple data streams), MISD (multiple instruction streams, single data stream), and MIMD (multiple instruction streams, multiple data streams).

2.1 SISD

Conventional single processor computers are classified as SISD systems. Each arithmetic instruction initiates an operation on a data item taken from a single stream of data elements.

2.2 SIMD

SIMD machines have one instruction-processing unit, sometimes called a controller, and several data processing units, processing elements. The control unit is responsible for fetching and interpreting instructions. When it encounters arithmetic or other data processing instruction, it broadcasts the instruction to all processing elements, which then all perform the same operation but on different data. To allow for needed flexibility in implementing algorithms, a processing element can be deactivated. Thus on each instruction, a processing element is either idle, in which case it does nothing, or it is active, in which case it performs the same operation as all other active processing elements.

Two possible SIMD organizations are possible. The first way is to view the structure of a SIND machine is as a set of N processing elements interconnected by a network and fed

instructions by the control unit. Each processing element consists of a processor with its own memory. This configuration is shown in figure 13, and is called processing-element-to-processing-element organization. The interconnection network is unidirectional and connects each processing element to all processing elements or to a subset of them.

The alternative SIMD-machine organization is to position the interconnection network between the processors and the memories. This kind of configuration shown in Figure 14 is called the processor-to-memory organization.

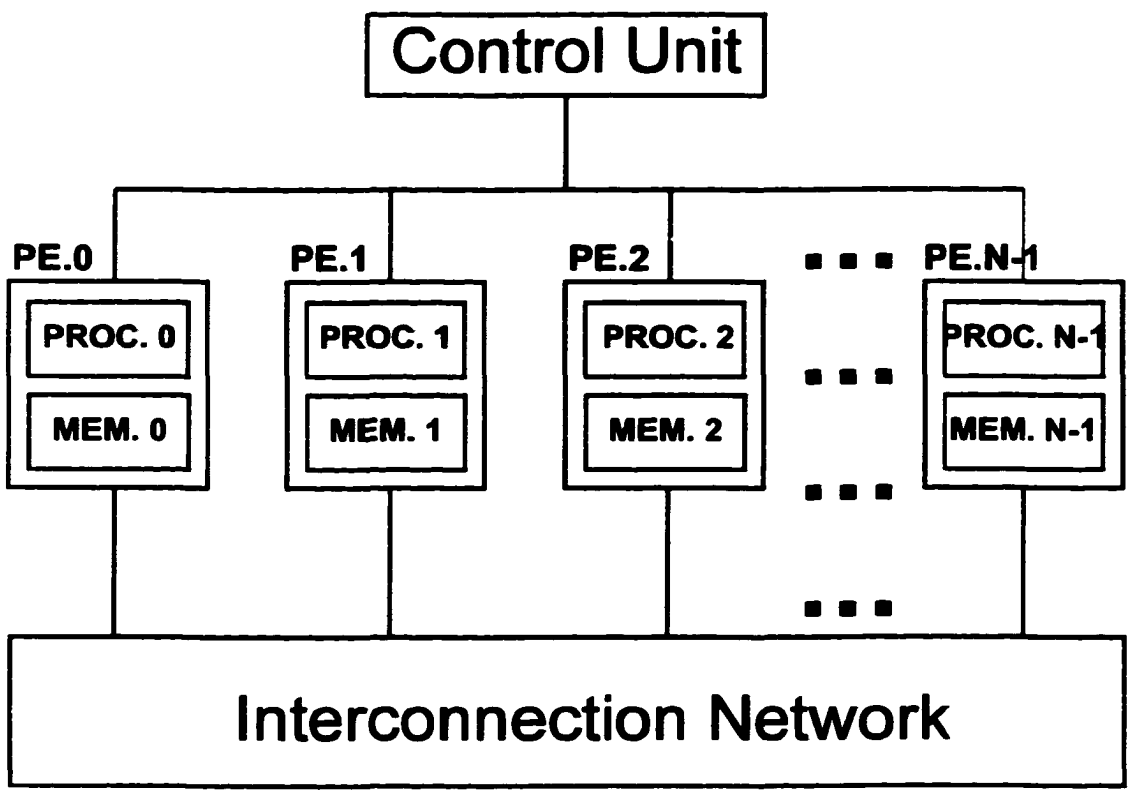


Figure 13: Processing-Element-to-processing-Element SIMD machine.

2.3 MISD

There are few machines in this category, none that have been commercially successful or had any impact on computational science.

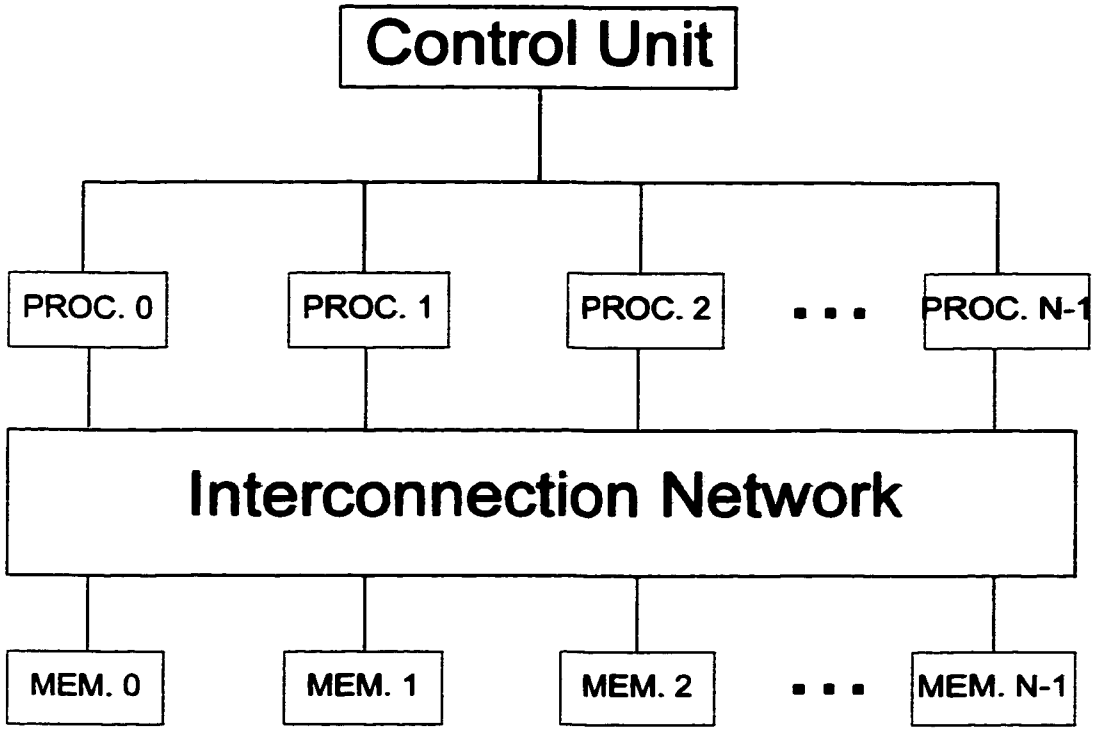


Figure 14: Processor-to-Memory SIMD machine.

2.4 MIMD

A MIMD machine consists of N processors, N memory modules, and an Interconnection network. Each of the N processors follows its own program, thus multiple instruction streams. Each processor fetches its own data, thus multiple data stream. The interconnection network provides a means of communication among the processors and memory modules.

A MIMD machine includes a collection of independent processors that can be used together to do tasks faster than a single processor. The processors in a MIMD system operate asynchronously with respect to one another unlike the synchronous operation of all active processors in s SIMD machine.

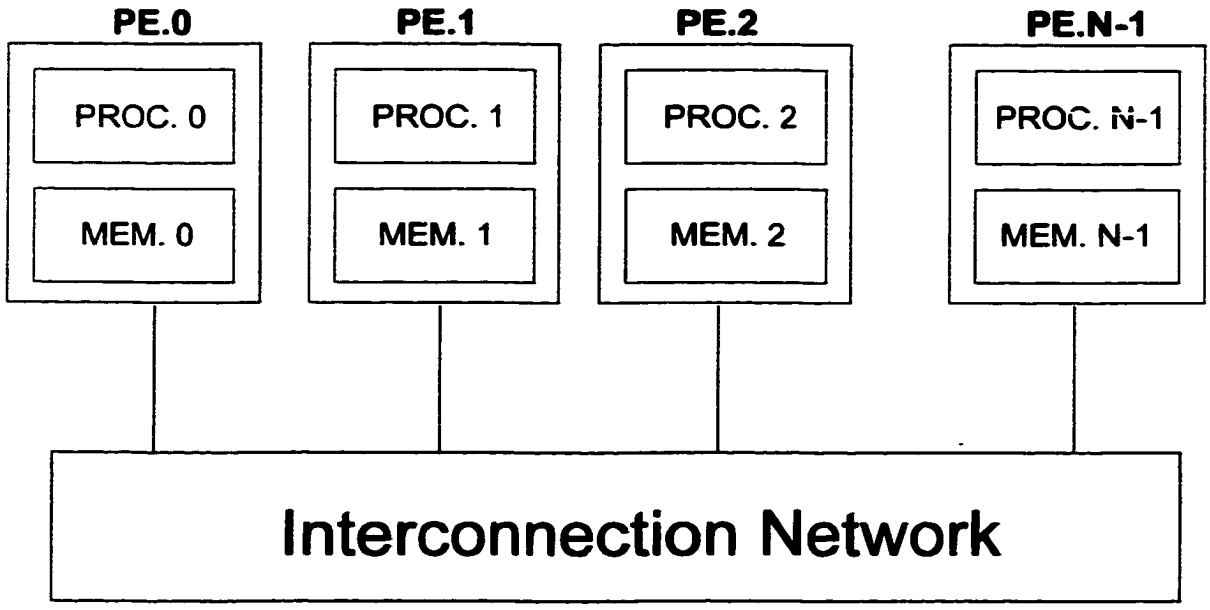


Figure 15: Processing-Element-to-processing-Element MIMD machine.

Two MIMD configurations are possible just as in the case of SIMD: processing-element-to-processing-element organization and the processor-to-memory organization. Figure 15 and Figure 16 shows the two configurations respectively.

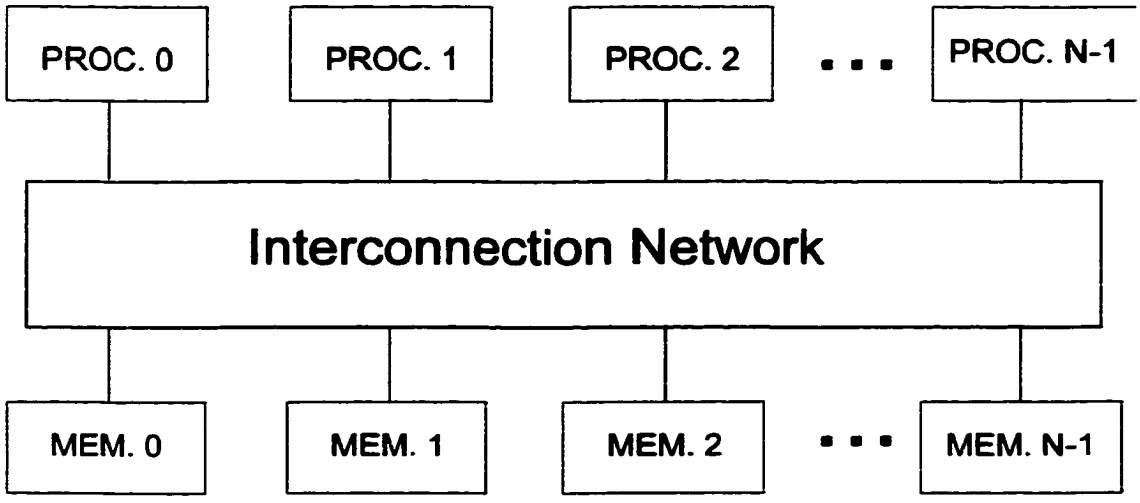


Figure 16: Processor-to-Memory MIMD machine.

The interconnection network that is supporting the communication inn either above described systems must be as powerful, flexible, reliable, and optimal as possible. That is of course taking in account the cost of the interconnection network. Designing very good interconnection network is not the only problem, economics plays a big role in the field of interconnection network.

3 Problems related to Interconnection Network.

Following is a list of most important problems that should be addressed in the design of any interconnection networks.

Labeling: By labeling we mean the names of the nodes or the addresses of the processing elements in the interconnection network. Labels must be unique. Good, well-defined labeling can ease the task of connectivity. It could also simplify the study of properties of the interconnection network such as its routing properties.

Connectivity: Connectivity of an interconnection network means that there is a possible path between each two nodes of the network. In the case when real directed graphs we can distinguish between weekly and strongly connected networks. A network is strongly connected if and only if an edge from A to B implies that there is an edge from B to A. A network is weakly connected if there is a path between any two nodes in the networks but not necessarily in both directions.

Point to Point routing: This describes the path that we have to take in order to make a transfer between given source and destination. It should the minimal length path between

given nodes. In case of unavailability of the path, an alternative path should be considered. Node independent routing functions are of special interest since the routing functions can be applied at any node.

Alternative and Parallel Paths: In case of unavailability of a path, an alternative path must be considered. The alternative path should also be minimal, but that is not possible all the time. The unavailability of the path might be due to a faulty link, or to the malfunction of a switch, a router, a node, ... In this case. The alternative path and the original path should be disjoint, that is, all nodes in the alternative path should be different except for the source and the destination. Such paths are also called parallel.

Off-line routing: We talk about off-line routing or static routing when we have global information about all possible destinations and therefore we could construct tables specifying the routes from any nodes to any destination. This scheme although easy and efficient, requires some overhead such as keeping track of tables, and handicaps the system from dynamic evolution such as expansion, or simulation of other networks. Also, in case of the failure of a node, the tables no more contain correct information, the network is no more functional. The off-line routing needs to be redone.

On-line routing: The alternative to off-line routing. It is also referred to a dynamic routing. Since the information about all the network members might not be present at start, or might change dynamically, we can no longer predetermine all the routes from any node to any destination. For a message to go from source A to destination B, the route is determined along the way as the message is travelling, or the route is dynamically calculated given the source address, the destination address and some sort of local

information. The advantage of this kind of routing is its flexibility and its partial independence from the state of the network.

Broadcasting: Broadcasting is a routing policy in which data from a single node is sent to more than one node. The following are the most common forms of broadcasting:

Single-node broadcast: A case when a node is sending data to all other nodes.

Single-node scatter. When a node is sending a different packet to every node in the network.

Multi-node broadcast. When all nodes perform their own broadcasts simultaneously.

Total exchange. When all nodes perform their scatters simultaneously.

Embedding: Interconnection networks are usually viewed as connected graphs with nodes representing the processing elements and vertices defining the physical links between the processing elements. Embedding refers to the fact that an interconnection network topology is a sub-topology of another interconnection network. In other words, Let G_1 be the graph that corresponds interconnection network1 and G_2 the graph that corresponds interconnection network2. If G_1 is a sub-graph of G_2, we say that interconnection network1 can be embedded in interconnection network2. Embedding is closely related to simulating interconnection network.

Simulation: Since many types of interconnection network exist and are in use, there is a problem of portability of parallel programs, i.e. how to run a parallel programs, written for a particular topology, an interconnection network with different topology. The usual way to solve this is to simulate one topology with the other. The task of using interconnection network1 to perform the connections that are part of interconnection

network2 is referred to as simulating interconnection network2 with interconnection network1.

Partitioning: The partionability of an interconnection network is the ability to divide the network into independent sub-networks of different sizes. Each sub-network of size N' < N must have all the interconnection capabilities of a complete network of that same type built to be of size N'.

4 Formal Modeling Techniques

Several mathematical models were presented for Interconnection networks. No single model presents a standard and efficient modeling technique that could be applied to most or all classes of networks. Most existing modeling techniques are structure dependent and as a result they are static and are only good to study the static properties of networks. Once the structure of the network is established, these properties do not change. These models do not if at all support the capability of investigating of the dynamic properties of networks.

Other models have support for the investigation of the dynamic properties of Interconnection Networks but they only apply to a class of network and therefore are not standard and can not be used to model any networks that fall outside of the supported class. By dynamic properties we mean those properties that can be modified while the network is operational. It includes characteristics like different types of routing or emulation of other topologies in the given network.

Finally, some models such as the finite state automata model, support both static and dynamic capabilities of a wide range of Interconnection Networks, but still have certain drawbacks such as a limitation on the size of the network. These models do not yet present a unified way to study the properties, static and dynamic, of most interconnection networks.

In this section we will present a review of the most common mathematical models and point out some of their drawbacks. At the end of this chapter a small discussion will be presented to define what would be a good modeling technique. In chapter 2, a new modeling technique will be introduced and in chapter 3 the modeling technique will be partially modified and extended and its potential will be fully explored in that chapter and in subsequent chapters.

4.1 Interconnection functions

An interconnection network can be described by a set of interconnection functions [25]. Each interconnection function is a bijection (permutation) on the set of PE addresses. When an interconnection function is applied at a PE_i, a send operation is performed from PE_i to PE_j such that $f(PE_i) = PE_j$. For example let f be defined over the set N = {0,1, 2, 3...7}, the set of nodes, n be the number of nodes is equal to |N| → n = 8 and f defined as follows: f(PE) = PE + 1 mod n. Therefore if the address of PE = 3, then f(PE) = PE + 1 mod 8 = 4 that is a transfer of data will occur between PE 3 and PE 4. If the address of PE = 7, then f(PE) = PE + 1 mod 8 = 0 that is a transfer of data will occur between PE 7 and PE 0. An interconnection function f mathematically maps the address of PE to the address of f(PE). We now formally define an interconnection function:

Definition 1.1: Let N be a node set such that N = {0,1 ... n}. An interconnection function is a mapping from N into N, i.e. f: N→N.

An interconnection network can be defined as a pair (N, F) where F is a set of interconnection functions having the following property:

Identity map i defined as $i(x) = x, \forall x \in N$ is in F.

For any two x, y ∈ N, there is a finite sequence $f_1, f_2 \ldots f_k$ of function from F such that $f_1 \circ f_2 \circ \ldots \circ f_k(x) = y$, where ° stands for composition

Nodes x, y ∈ N are directly connected iff there exists f ∈ F such that f(x) = y or f(y) = x.

Example: The n-cube's can be modeled using the interconnection function:

$Cube_i (a_{r-1} a_{r-2} \ldots a_{i-1}, a_i, a_{i+1} \ldots a_1 a_0) = a_{r-1} a_{r-2} \ldots a_{i-1} a_i' a_{i+1} \ldots a_1 a_0$ where $0 \leq i < r$.

Using the above interconnection function, given the node set we can find all the neighbors for all the elements of the node set and therefore defining the edge set of the network. For illustration, let the node set of a 3-cube N = {000, 001, 010, 011, 100, 101, 110, 111}. Thus F = $\{f_1, f_2, f_3\}$ where $f_1 = cube_0 (a_2a_1a_0) = a_2a_1a_0'$, $f_2 = cube_1 (a_2a_1a_0) = a_2a_1'a_0$, $f_3 = cube_2 (a_2a_1a_0) = a_2'a_1a_0$, where a_i' denotes the complement of a_i.

$cube_0 (000) = 001$, $cube_1 (000) = 010$, $cube_2 (000) = 100$.

$cube_0 (001) = 000$, $cube_1 (001) = 011$, $cube_1 (001) = 101$.

$cube_0$ (010) = 011, $cube_1$ (010) = 000, $cube_2$ (010) = 110.

$cube_0$ (011) = 010, $cube_1$ (011) = 001, $cube_2$ (011) = 111.

$cube_0$ (100) = 101, $cube_1$ (100) = 110, $cube_2$ (100) = 000.

$cube_0$ (101) = 100, $cube_1$ (101) = 111, $cube_1$ (101) = 001.

$cube_0$ (110) = 111, $cube_1$ (110) = 100, $cube_2$ (110) = 010.

$cube_0$ (111) = 110, $cube_1$ (111) = 101, $cube_2$ (111) = 011.

The edge, E, set is the union of all above edges. We therefore get:

E = {(000, 001), (000, 010), (000, 100), (001, 011), (001, 101), (010, 011), (010, 110), (011, 111), (100, 101), (100, 110), (101, 111), (110, 111)}. See Figure 17.

The problem with this approach is that it gives a narrow view of the system. No well-

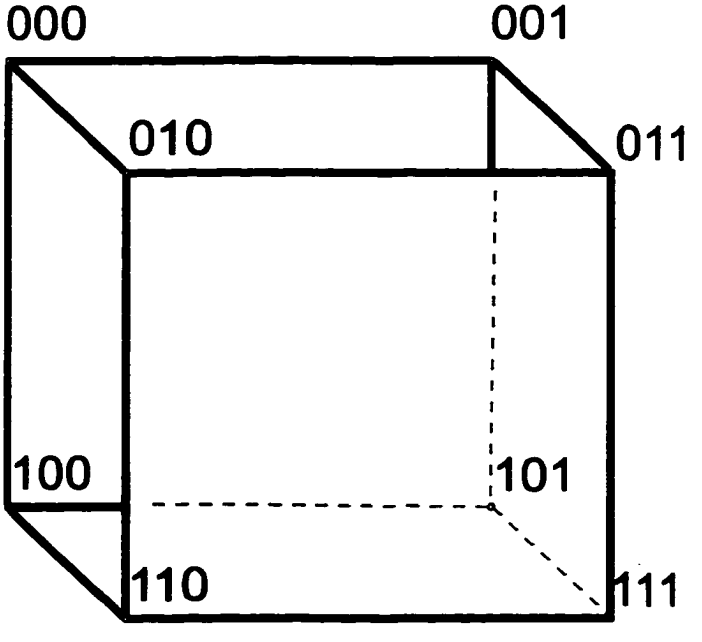


Figure 17: A 3-cube with edges generated by the interconnection functions.

defined methodology for investigating the properties of the network is given. Dynamic properties in particular are difficult to discover and study.

4.2 Networks as Permutations.

This modeling technique is an extension of interconnection functions [25].

A permutation is simply a bijection (one-to-one onto) from a set into itself as is the case for Interconnection functions [25]. A cyclic notation is used to represent an interconnection function f as a permutation. A permutation can be represented as a cycle (a b c...g h), or as a product of cycles (a...e) (g...h) [25, 14]. The cycle (a b c...g h) means that f(a) = b, f(b) = c, ..., f(g) = h and f(h) = a. What this really means is that a is physically connected to b, b physically connected to c and so on. Some interconnection functions create two or more sets of independent cycles as is the case with PM2I interconnection network with i = 1 and N = 8, in which case we get 0 is connected to 2, 2 to 4, 4 to 6 and 6 to 0. On the other hand get 1 is connected to 3, 3 to 5, 5 to 7 and 7 to 1. For N = 8, we write $PM2_{+1}(P) = P + 2^I = P + 2 = (0\ 2\ 4\ 6)\ (1\ 3\ 5\ 7)$ which is the product of 2 cycles.

Given a set of nodes on which two permutations p and q are defined, the product of the two permutation i.e. pq is defined as first applying p to every element in the node set then applying q. Let p = (0 1 2 3 4 5 6 7) and q = (0 2 4 6) (1 3 5 7) then the product of p and q leads to pq = (0 3 6 1 4 7 2 5) because q(p(0)) = 3, q(p(3)) = 6 and so forth.

This technique has the same semantics as interconnection functions but provides a very different algebra, which could be used as an alternative to investigate the properties of an interconnection network from a different angle.

4.3 Networks as Algebraic Structures.

This model [24, 2] is applied to cases in which the graph of the network can be represented as a graph of an algebraic structure. We define here algebraic structure A as a groupoid, that is a finite set with a defined binary operation A = (A, *) where * denotes the operation. A generating set S for *A* is a subset of A (without identity element, if there is one) if any element of A can be expressed (not necessarily uniquely) as a product of elements in S. We denote this as A = *gp(S)*. The graph of the structure A with respect to S is *F(A, S)* = (A, *E)* where the set of edges E is defined as follows: There is an edge between a, b $\in$A if and only if for some s $\in$ S.

The most interesting structures are semigroups, monoids, groups and with additional operation even vector spaces. In the case of groups the underlying networks are called Cayley *networks* or graphs. So in a case of a Cayley graphs the vertices correspond to the elements of the group and edges depict actions of the generators [14, 2]. All Cayley graphs are vertex and edge symmetric.

Example: n-cube is a Cayley graph.

A 3-cube can be described as a group graph built from the generating set made of the following transpositions $C_3 = \{s_1 = (12), s_2 = (34), s_3 = (56)\}$. The 8 elements of the 3-cube can be obtained by actions of C_3 on the identity permutation of 6 symbols (123456).

4.4 Networks as Finite Automata

In this model, the network is represented as finite state automata. Nodes of the network are the states and strings accepted by the finite state automata describe possible path. Any node (state) in the network is starting and finite state at the same time. We can use any finite alphabet for the set of inputs, but it is more convenient to use the same alphabet used for labeling the nodes.

We can formally define interconnection networks in this model as follows:

Definition 1.2: An interconnection network modeled as a finite state automaton of rank k is a triple (X, C, ϕ):

X is a nonempty set of N states (N nodes): 0...N-1

C is The control set: A control set is the set of control vectors. A control vector is a binary n-vector that can be used as a manipulative tool in defining nodes, which are directly linked together.

ϕ is a mapping from the product set X x C onto X and is referred as the transition function, or the routing function.

Example: Automata model of ring networks.

An n-ring network is consists of N nodes with the node set $X = \{0, 1,..., N-1\}$. The control set $C = \{1\}$, in case the ring is unidirectional, and $C = \{-1, 1\}$ in case the ring is bi-directional. The ring routing function is defined as $\phi(x, c) = x + c \pmod N$ $\forall c \in C$.

A finite state automata model of a network (X_n, C_n, ϕ) is said to be bit-controlled if and only if $\phi(x, c) = x^c = x +_2 c$ for all $x \in X_n$, and all $c \in C_n$.

Example: Modeling the class of Hypercubes.

The n-Hypercube is a bit controlled type of network with $N = 2^n$ nodes. The control set contains n elements represented as all n-tuples made of one symbol 1 and n-1 symbols 0, that is $C = \{2^i / \ 0 \leq i \leq n-1\}$ and in binary form $C_n = \{100....0, 010..0, ..., 000...01\}$.

More details about networks as finite automata model, and control sequences can be found in [18, 4].

Chapter 2

Historical background

In a recent publication [23], a new class of interconnection network topologies, the class of linear recursive networks, was introduced. Some well-known member of this class includes the Hypercube and the generalized Fibonacci cube. Several static as well as dynamic properties of linear recursive networks, such as diameter and node degree, were studied in.

This chapter reports the results developed in [23], and motivates a unified and more general approach to study the static and dynamic properties of members of the class of linear recursive networks. This new approach will be presented in the next chapter. We will briefly review the Fibonacci cube reported in [16] and the k-ary n-cube in[17].

First we will review properties of linear recurrence relations that are relevant to our discussion, and give several results that will be used throughout the thesis.

1 A Brief Review of Linear Recurrence Relations.

Definition 2.1: A recurrence relation is an equation that allows us to compute the n^{th} term of a sequence $<A_n>$ from the preceding terms (A_{n-1}, A_{n-2}, ...), i.e., $A_n = f_1(n) A_{n-1} + f_2(n) A_{n-2} + \ldots + f_k(n) A_{n-k} + D(n)$ where for all i, $f_i(n)$ and $D(n)$ are functions of integer n.

Definition 2.2: A recurrence equation in which D(n) = 0 is called a *homogenous* recurrence equation.

Definition 2.3: A recurrence equation in which all f_i's are constant is called a constant coefficient recurrence equation.

Definition 2.4: A recurrence equation is called an r^{th}-order recurrence if it allows us to compute A_n from the preceding r terms, namely A_{n-1}, A_{n-2} ...A_{n-r}.

Definition 2.5: A linear recurrence equation is a recurrence equation of the form:

$$A_n = b_1(n)\, A_{n-1} + b_2(n)\, A_{n-2} + \ldots. + b_0(n)\, A_0.$$

The above recurrence equation is called linear because every occurrence of A_i is to the first power.

Note: Throughout this thesis, we will refer to a Linear Homogeneous Recurrence equation with Constant Coefficients as Linear Recurrence Equation for short. We will specifically specify otherwise.

For example, $C_n = C_{n-1} + n\text{-}1$ is a first-order linear recurrence equation. $S_n = 2\, S_{n-1}$ is also a first-order linear recurrence equation. $B_n = B_{n-1} + B_{n-2}$ is a second-order linear recurrence equation. Finally, $K_n = K_{n-1}^2$ is a first-order non-linear recurrence equation.

1.1 Some results for Linear Recurrence Equations.

Theorem 2.1: If $A_n = b\ A_{n-1}$ for all $n > m$ and $A_m = c$ then $A_n = c\ b^{n-m}\ \forall\ n > m$.

For example, Given the set B={1 ... n}. Find the number of subsets of B.

Lets S_n denotes the number of subsets of the set B with n elements. Obviously $S_0 = 1$, since the empty set only has itself for subset.

$S_n = 2\ S_{n-1}$, then $S_n = 1.2^{n-0} = 2^n$.

In General, $S_n = b\ S_{n-1} = b\ (b\ S_{n-2}) = \ldots = b^n\ S_0$. To relate to the theorem2.1 set m to zero. So, $S_n = c\ b^{n-m} = S_0\ b^{n-0} = b^n\ S_0$.

Theorem 2.2: The sequence $A_n = r^n$ is a non-zero solution to the recurrence relation $A_n + b\ A_{n-1} + c\ A_{n-2} = 0$ if and only if r is the root of the polynomial $x^2 + b\ x + c = 0$.

Proof:

if $A_n = r^n$ is a solution to $A_n + b\ A_{n-1} + c\ A_{n-2} = 0$, substitute r^n in the recurrence relation to obtain $r^n + b\ r^{n-1} + c\ r^{n-2} = 0$. Divide this nth degree equation by r^{n-2} and get $r^2 - b * r - c = 0$. Therefore r is a root of $x^2 + b\ x + c = 0$.

Theorem 2.3: If the equation $x^2 + b\ x + c = 0$ has two different roots, r_1 and r_2, then each solution to the recurrence relation $A_n + b\ A_{n-1} + c\ A_{n-2} = 0$ may be written as $c_1\ r_1^n + c_2\ r_2^n$ for some choice of c_1 and c_2.

2 Introduction to Linear Recursive Networks

This section summarizes the work presented in [23]. It introduces the class of Linear Recursive networks and briefly discusses their static and dynamic properties.

Definition 2.6: An n-dimensional linear recursive network, or LRN for short, is associated with a linear recurrence of the form:

$$X_n = a_1 X_{n-1} + a_2 X_{n-2} + \ldots + a_k X_{n-k},$$

where a_i, $1 \leq i \leq k$, are non-negative integer such and $a_k \neq 0$.

In definition 2.6, subscript n denotes the dimension of the LRN while the parameter a_i represents the number of occurrences of a lower dimension network X_{n-1}, i.e. the (n-1)-dimensional network, within the n-dimensional network X_n. X_n itself denotes the size of the n-dimensional network.

Example 2.1:

The recursion $X_n = 2\ X_{n-1}$ would correspond to an n-cube since each n-dimensional Hypercube consists of two copies of (n-1)-dimensional Hypercubes. [23]

Example 2.2:

The recursion $X_n = X_{n-1} + X_{n-2}$ corresponds to the Fibonacci cube [23] since each n-dimensional Fibonacci cube consists of an (n-1)-dimensional Fibonacci cube and (n-2)-dimensional Fibonacci cube.

Definition 2.7: Each LRN $X_n = a_1 X_{n-1} + a_2 X_{n-2}+\ldots+ a_k X_{n-k}$ is associated with a generating sequence $A = a_1 a_2 \ldots a_{n-k}$, or simply the generator A.

For example the generating sequence of the Hypercube is A = 2, and the generating sequence for the Fibonacci-cube is A = 11.

Definition 2.8: We define Weight (A) to be the number of non zero bits in A and J to be the set of all i such that $a_i \neq 0$, i.e. $J = \{i \mid a_i \neq 0\}$.

Corollary 2.1: |J| = Weight (A).

Proof:

Follows directly from the fact that J is the set of all $a_i \in A$, such that $a_i \neq 0$.

For example if A= 1001, Weight (A) = 2, J = {1, 4} and Weight (A) = |J|.

Definition 2.9: The seed string A_j defined over J, for each non-zero bit a_j of A, is made of the prefix of A, excluding a_j, concatenated with 0. That is $A_j = a_1.a_2...a_{j-1}$. 0: $j \in J$.

Corollary 2.2: There exist exactly Weight (A) seed strings.

Proof:

To each non-zero a_i corresponds a seed string A_i. Therefore the number of seed strings is equal to the number of non-zero a_i that is |J| = Weight (A).

Example 2.3:

Let A=1001. Then Weight (A) = 2. Thus J = {1, 4}. Therefore the two seed strings A_1 and A_4 are:

$A_1 = 0$ and $A_4 = 1000$.

Example 2.4:

Let A=1101011. Then Weight (A) = 4. Thus J = {1, 2, 4, 6, 7}. Therefore the four seed strings A_1, A_2, A_4, A_6 and A_7 are:

$A_1 = 0$, $A_2 = 10$, $A_4 = 1100$, $A_6 = 110100$ and $A_7 = 1101010$

Now we are ready to define the Graph of LRNs. We will use the term hamming distance between x and y, H(x, y), to denote the number of bits where the strings x and y differ.

Definition 2.10: An LRN is the graph $G_n(A) = (V_n(A), E_n(A))$. Where $V_n(A)$ is the set of vertices, nodes, of the graph and $E_n(A)$ is the set of edges, physical links between the nodes. $V_n(A)$ is defined as follows:

$V_0(A) = \{e\}$, $V_n(A) = \bigvee_{j \in J} A_j \,.\, G_{n-j}(A)$ and for $\forall\, n < j$ $A_j \,.\, G_{n-j}(A) = \{a_1\, a_2 \ldots a_n\}$. $E_n(A) =$ $\{(x, y) |\, x, y \in V_n(A)$ and $H(x, y) = 1\}$. e denotes the empty string.

Example 2.5:

We will take a generator A = 10101 and generate $V_n(A)$, $E_n(A)$ then draw $G_n(A)$ for n=0, 1, 2, 3, 4, 5.

A = 10101. Then Weight (A) = 3 =|J|, where J = {1, 3, 5}.

Therefore, there are three seed strings A_1, A_3 and A_5:

$A_1 = 0$, $A_3 = 100$, $A_5 = 10100$.

$G_n(A) = (V_n(A), E_n(A))$.

$V_n(A) = \bigvee_{j \in J} A_j \,.\, G_{n-j}(A) = A_1 \,.\, G_{n-1} + A_3 \,.\, G_{n-3} + A_5 \,.\, G_{n-5}$.

$V_0(A) = \{e\}$.

$E_0(A) = \{\}$

$V_1(A) = A_1 \,.\, G_{1\text{-}1} + A_3 \,.\, G_{1\text{-}3} + A_5 \,.\, G_{1\text{-}5} = \{0\} \cup \{1\} \cup \{1\} = \{0, 1\}$

$E_1(A) = \{(0, 1)\}$

$V_2(A) = A_1 \,.\, G_{2\text{-}1} + A_3 \,.\, G_{2\text{-}3} + A_5 \,.\, G_{2\text{-}5} = \{00,01\} \cup \{10\} \cup \{10\} = \{00,01,10\}$

$E_2(A) = \{(00, 01), (00, 10)\}$

$V_3(A) = A_1 \,.\, G_{3\text{-}1} + A_3 \,.\, G_{3\text{-}3} + A_5 \,.\, G_{3\text{-}5} = \{000,001,010\} \cup \{100\} \cup \{101\}$

$= \{000,001,010,100,101\}$

$E_3(A) = \{(000, 001), (000, 010), (000, 100), (001, 101), (100, 101)\}$

$V_4(A) = A_1 \,.\, G_{4\text{-}1} + A_3 \,.\, G_{4\text{-}3} + A_5 \,.\, G_{4\text{-}5} = \{0000,0001,0010,0100,0101\} \cup$

$\{1000,1001\} \cup \{1010\} = \{0000,0001,0010,0100,0101,1000,1001, 1010\}$

$E_4(A) = \{(0000, 0001), (0000, 0010), (0000, 0100), (0000, 1000), (0001, 0101),$

$(0001, 1001), (0010, 1010), (0100, 0101), (1000, 1001), (1000, 1010)\}$

$V_5(A) = A_1 \, G_{5\text{-}1} + A_3 \,.\, G_{5\text{-}3} + A_5 \,.\, G_{5\text{-}5} =$

$= \{00000,00001,00010,00100,00101,01000,01001,01010\} \cup$

$\{10000,10001,10010\} \cup \{10100\}$

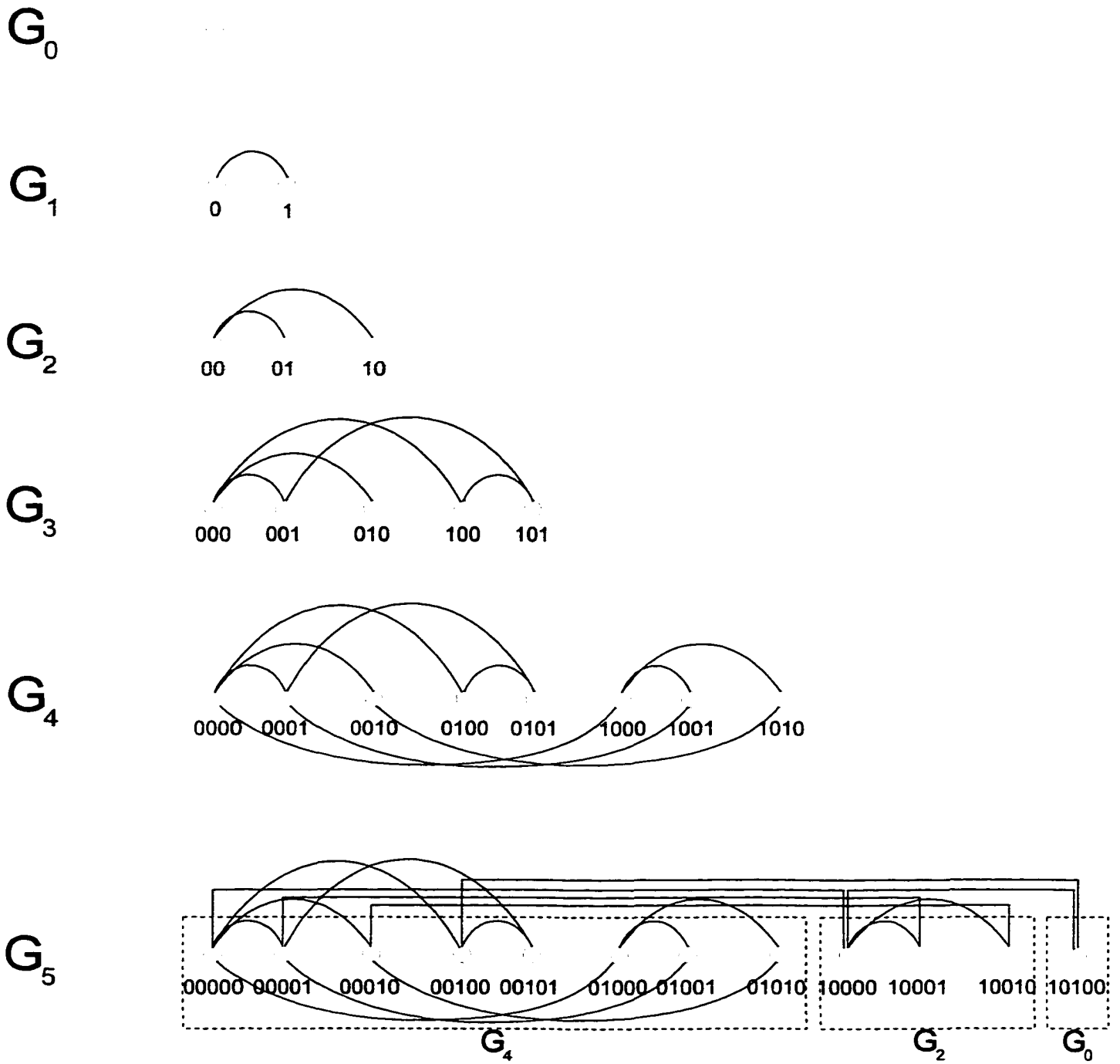


Figure 18: Linear Recursive Network generated by A = 10101.

={00000,00001,00010,00100,00101,01000,01001,01010,

10000,10001,10010,10100}

$E_5(A)$ = {(00000, 00001), (00000, 00010), (00000, 00100), (00000, 01000),

(00001, 00101), (00001, 00101), (00001, 01001), (00001, 10001),

(00010, 01010), (00010, 10010), (00100, 00101), (00100, 10100),

(01000, 01001), (01000, 01010), (10000, 10001), (10000, 10010),

(10000, 10100)}

Figure 18 shows the Graphs G_0, G_1, G_2, G_3, G_4 and G_5. We see that G_5 contains one copy of G_4, one copy of G_2 and one copy ofG_0.

Definition 2.10 gives a well-defined method to derive the graph $G_n(A)$ of the network, i.e. the node set $V_n(A)$ and $E_n(A)$. This method will be expanded upon, in the next chapter, and will be used to study the static and dynamic properties of any LRN.

2.1 Properties of Linear Recursive Networks.

Several basic properties of LRNs, such as node degree, number of edges, diameter, are presented in [23]. We denote the size (number of nodes) of the network by $N = X_n$.

Node degree: The degree of nodes in an LRN is bounded by a logarithmic function of N.

Number of edges: The total number of links (edges) is bounded by O(N log N). The multiplier is a constant characteristic of the generator n.

Diameter: The diameter of an LRN is bounded by above by O(Log N). This implies that any two arbitrary nodes in a LRN can communicate by going over no more than O(log N) intermediate nodes.

Several interesting properties were presented in [23] such as the decomposition of $G_n(A)$ and the upper bound of the distance between any two nodes.

Decomposition: Let $G_n(A)$ be the graph of an LRN. Then for all $n \geq |A|$, $G_n(A)$ can be decomposed to m sub-graphs $g_1, g_2, \ldots, g_m$ such that $g_i \equiv G_{n-j}(A)$ for $1 \leq i \leq m$ and $g_k \cap g_l = \varnothing$ for $k \neq l$.

Upper bound for the distance: Let D(x, y) denote the distance between two nodes x and y in $G_n(A)$, an LRN. Then $D(x, y) \leq 2n$.

3 The Fibonacci Graphs

In [16], Fibonacci graphs are modeled using the techniques presented in[23]. A Fibonacci cube is defined recursively as an LRN, and some of its properties are studied. Routing strategies for Fibonacci graphs are developed. This section will summarize the work presented in [16].

Definition 2.11: The Fibonacci Graph (FG) of size n and generator sequence $A = a_1a_2$ $G_n(A)$, where $a_1 > 0$, and $a_2 > 0$, is the graph with the vertex (or node) set $V_n(A)$ such that:

$V_0(A) = \{e\}$

$V_1(A) = A_1$

$V_n(A) = A_1 \circ V_{n-1}(A) \cup A_2 \circ V_{n-2}(A)$, for all $n \geq 2$.

Two nodes X and Y of $G_n(A)$ are connected if and only if $H(X, Y) = 1$.

3.1 Topological properties of Fibonacci Graphs

Definition 2.12: Two generator sequences A and A' are equivalent, denoted $A \approx A'$, if and only if $G_n(A)$ and $G_n(A')$ are isomorphic images of each other, for every n greater than zero.

Theorem 2.4: For $a_1 > 0$ and $a_2 > 0$, let $A = a_1a_2$ be a generator string, then, for any integer $b \leq a_1$, $A \approx (b_1a_20) \oplus (0[(a_1 - b_1) \times a_1][(a_1 - b_1) \times a_2])$, where $-$ and $\times$ denote the standard subtraction and multiplication, and $\oplus$ denotes the xor binary operation.

Theorem 2.5: For $a_1 > 0$ and $a_2 > 0$, let $A = (a_1) \circ (a_2)$ be a generator of a Fibonacci graph $G_n(A)$, then:

$A \approx B_0 \approx B_1 \approx \ldots \approx B_I \approx B_{I+1} \approx \ldots$ where, $_0 = b_0b_1 = A$ and for all $i \geq 0$, $B_I = (0^i) \circ (b_i) \circ (b_{i+1}) \circ (b_{i+2}) \circ \ldots$ and B_{I+1} is a string satisfying the recurrence relation:

$B_{I+1} = [(b_i) \times E_i (B_i)] \oplus [(0^{i+1}) \circ (b_{i+1}) \circ (b_{i+2}) \circ \ldots]$

The number of subgraphs isomorphic to $G_{n-j}(A)$ is the digit b_j of the string B_{j-1} on part (1)

Lemma 2.1:

For all $n \geq 0$, $0^n \in V_n(A)$.

Fibonacci graphs are connected.

The diameter of $G_n(A)$ is bounded by 2n.

Every node of $G_n(A)$ has degree no more than (r-1)n.

Routing algorithms in Fibonacci Cubes were developed. For more details on Fibonacci graphs and their static and dynamic properties, refer to [16].

4 K-ary n-cube Network: n:k cube

Definition 2.13: The n:k cube Q(n, k) consists of $N = k^n$ nodes. Each node X is labeled by a sequence of n digits, i.e. $X = x_{n-1}x_{n-2}\ldots x_0$, where $0 \leq x_i \leq k-1$, for all $0 \leq i < n$.

Figure 19: 8-ary 1–cube: 1:8 cube.

The N nodes of an n:k cube are arranged in n dimensions, with k nodes per dimension: n is the dimensionality and k is the radix. Figure 19 an n:k cubes.

An n:k cube can be decomposed into k (n-1):k cubes or k^2 (n-2):k cubes or in general k^β (n-β):k cubes for all $\beta \leq n$. The relation between Q(n, k) and its components can be described by the following recurrence relations: $Q(n, k) = k\ Q(n - 1, k)$ or $Q(n, k) = k^2\ Q(n - 2, k)$ or in general $Q(n, k) = k^\beta\ Q(n - \beta, k)$. These equations are refereed to as characteristic equations of n:k cubes. Hence, the n:k cube is a Linear Recursive network.

With further algebraic manipulations, they showed that n:k cubes are generated by the generator $A = (k - 1) \ldots (k - 1)k = (k - 1)^\beta k$, where $(k - 1)^\beta$ denotes the concatenation of (k - 1) done β times.

For more details about the n:k cubes and their modeling as Linear Recursive Networks, refer to [17].

5 Projections

The methodology presented in [23] is a new method to be used in investigating the static and dynamic properties of an interconnection network. A LRN is an interconnection network whose size is modeled by a recurrence equation. That LRN is generated by the sequence of coefficients in the recurrence equation. But the topology of the network, (the way the nodes in the network are interconnected) does not depend on network's size. For the same number of nodes, we can obtain a variety of topologies, depending on how we define the edge set.

Here is a tentative (though not necessarily complete) list of problems and week points of the work done, sofar, on Linear recursive Networks:

1) The alphabet was restricted to the binary alphabet: The binary alphabet is widely used to generate node labels, but there exists Interconnection Networks that use different alphabet. For example, the n-star labels its node by generating all the combination of any given alphabet. The 3! nodes set of a 3-star network are {123, 132, 213, 231, 312, 321}.

2) Only Recursive Interconnections Networks with constant generators were considered: There exists LRNs whose sizes grow recursively, but the coefficient controlling the recursion is not constant. For example, a n-star graph contains n! nodes. Therefore, a 3-star graph contains 3! nodes, a 4-star graph contains 4! nodes , and 5-star graph contains 5! Nodes. In general, the n-star can be modeled as follows: $X_n = f(n)\, X_{n-1}$ where $f(n) = n-1$.

3) LRN were defined as those Interconnection Networks whose sizes grow recursively and whose edge set is $E=\{(a, b)|\ a, b \in V_n, \ \&\ H(a,b) = 1\}$: The recursion does not dictate the edge set but, it is used only for the purpose of node generation and control of the growth of the LRN. Restricting interconnection to hamming distance removes all the possible advantages that this new modeling technique brings about. There exist many Linear Recursive Networks that use different means to interconnect their nodes. For example, the n-star Interconnection Network interconnects nodes if their node labels differ by a permutation. The edge is always defined so that to satisfy certain requirement, application, and/or constraint. For example, by examining $X_n = 2\, X_{n-1}$, we only

understand that the number of nodes in a higher dimension graph is twice the size of the one of one lower dimension, but that does not dictate the topology. Examining the n-cube and the shuffle-exchange topologies supports the above argument. Both the n-cube and the shuffle-exchange graphs are made of 2^n nodes, but the graph look totally different as Figure 20 and Figure 21 shows.

4)

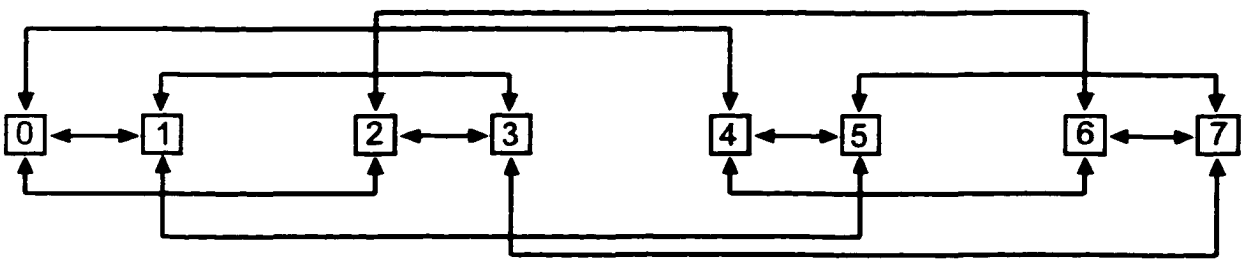


Figure 20: Single-stage implementation of a three dimensional Hypercube.

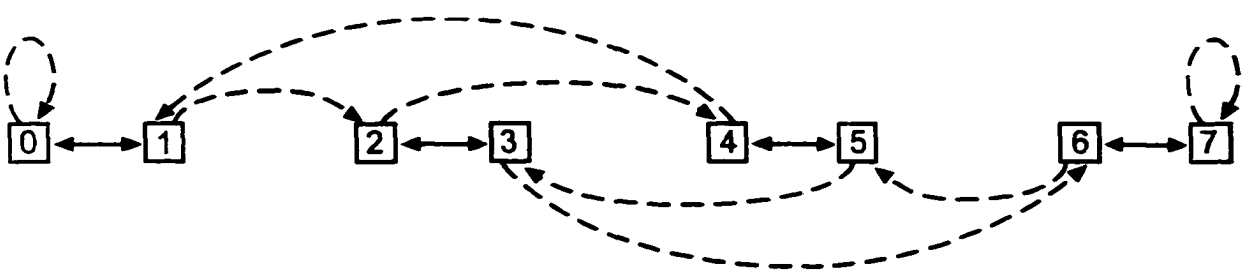


Figure 21: Single-stage implementation of a Shuffle-Exchange Network for N = 8.

5) Lack of new topologies, or enhancement of other existing topologies

6) Lack of efficient routing, embedding algorithms that apply to all or most of the elements of the Linear Recursive class of Interconnection Networks

We will first generalize the alphabet and rectify the definition as to separate between node set and edge set. The recursion will only be used to model the node set. The edge set will defined as to satisfy cost, diameter, parallel paths, ... We will develop general algorithms that apply to all the members of the class of Linear Recursive Networks. Static and dynamic properties of the networks will be linked to their generators.

The remainder of this thesis will cover the above mentioned points, and seek to take advantage of the modeling technique to participate in the improvement of the overall study and development of Interconnection Networks.

Chapter 3

Introduction to Linear Recursive Networks

The performance of a parallel processing system is greatly influenced by the topology of its interconnection network. The design of interconnection networks has been the focus of research in recent years. Interconnection networks must satisfy certain cost/performance constraints, and should contain a large number of static and dynamic properties. As the number of processing elements increases, the interconnection network has to scale as well, whiie maintaining its performance and efficiency. For example, the n-cube grows in each step by doubling its size. Otherwise, its routing algorithms will no longer be applicable and have to be replaced by more complex routing schemes. This restriction can sometimes be rather an inconvenience. Therefore, increasing the size of the network is costly and thus is unacceptable. Incomplete networks do not impose any restrictions on the size of the network. But incomplete networks usually require more complex routing and embedding algorithms than complete structures.

The size of the interconnection network, i.e., the number of nodes, can be defined recursively, so that the size of a large network is defined in terms of sizes of smaller networks. Networks would therefore scale in a well-defined fashion, and if the way networks scale is not acceptable, the recursive definition can be tweaked to suit the objectives of designers.

An interconnection network whose size is defined recursively is called a Linear Recursive Network.

1 Linear Recursive networks (LRN).

Definition 3.1: A LRN is an undirected graph $G_n = (V_n, E_n)$, where n is the minimum number of numerals which is necessary to label the nodes of G_n. V_n is the vertex set (node set) and is defined recursively. E_n is the edge set (the set of links between nodes). The size, of an LRN, X_n, satisfies the recurrence equation:

$$X_n = f_1(n)X_{n-1} + f_2(n)X_{n-2} + \ldots + f_k(n)X_{n-k}$$

where $f_i(n)$, $1 \le i \le k$, is a non-negative base-r integer function such and $f_k(n) \neq 0$.

Example 3.1: The n-cube C_n is a LRN. The size X_n of an n-cube satisfies the recurrence relation $X_n = 2\,X_{n-1}$.

Example 3.2: The n-star is a LRN whose size satisfies the recurrence relation $X_n = n\,X_{n-1}$

Definition 3.2: A LRN whose size is recursively defined by

$$X_n = f_1(n)X_{n-1} + f_2(n)X_{n-2} + \ldots + f_k(n)X_{n-k}$$

is said to be generated by the base-r sequence

$$A = f_1(n).f_2(n)\ldots f_k(n)$$

A is called the *generator* of the LRN.

Example 3.3: The size of an n-Cube, X_n, is described by $X_n = 2\ X_{n-1}$. Therefore, the generator o an n-cube is $A=2$.

Example 3.4: The size of an n-star is described by $X_n = n\ X_{n-1}$. Therefore, $A = n$.

Example 3.5: The size of a Fibonacci-cube is described by $X_n = X_{n-1} + X_{n-2}$. Therefore, its generator is $A=11$.

Definition 3.3: The number of non-zero base-r digits in A is called the Weight of A and is denoted by W(A).

Example 3.6: Consider a LRN which is generated by A=1001. Then, W(A)=2.

Definition 3.4: Let J be the set of all indices i such that $f_i(n) \neq 0$, i.e.

$$J = \{j_1, j_2, \ldots, j_m\} = \{i \mid f_i(n) \neq 0\}$$

Example 3.7: Let $X_n = X_{n-1} + 2X_{n-2} + 3X_{n-3}$. Then $A = 123$ and $W(A) = 3$. Therefore $J = \{1, 2, 3\}$, $|J|=3$.

Notations:

Following is a list of all notations used throughout the thesis.

Base-r numerals of a sequence with

r >10 are separated by dots. Otherwise, for r less than 10, no dots will be used and each numeral in the sequence is a distinguished component.

$y \circ \{x_1, x_2, \ldots, x_n\}$ denotes the concatenation of the string y with every member of the set $\{x_1, x_2, \ldots, x_n\}$, i.e., $y \circ \{x_1, x_2, \ldots, x_n\}=\{yx_1, yx_2,,\ldots, yx_n\}$.

$\{y_1, y_2, \ldots, y_n\} \circ \{x_1, x_2, \ldots, x_n\}$ denotes the concatenation of the sets $\{y_1, y_2, \ldots, y_n\}$ and $\{x_1, x_2,\ldots, x_n\}$, i.e., $\{y_1, \ldots, y_n\} \circ \{x_1, \ldots, x_n\} = \{y_1x_1,\ldots, y_1x_n, \ldots, y_n.x_1, \ldots, y_n.x_n\}$.

$\varnothing = \{\}$, and $\varnothing \circ A = A \circ \varnothing = A$

$|S|$ denotes the cardinality of the set S.

H(x, y) denotes the hamming distance between x and y. The hamming distance is the number of base-r numeral that are different in x and y. For example, if $x = x_0.x_1.x_2 = 1.0.1$, and $y = y_0.y_1.y_2 = 0.1.1$, the $H(x, y) = 2$ since $x_0 \neq y_0$ and $x_1 \neq y_1$.

Example 3.8: Let $X_n = X_{n-1} + 12X_{n-2} + 30X_{n-3} + 2X_{n-4}$ then *A= 1.12.30.2.* If we simply wrote *A=112302*, confusion could occur, and A could have been interpreted as being the generator of $X_n = X_{n-1} + 1X_{n-2} + 2X_{n-3} + 3X_{n-4} + 2X_{n-6}$.

Example 3.9: *1 ° {23, 34} = {123, 134} and 1 ° {12.3, 3.12} = {1.12.3, 1.1.12}.*

Example 3.10: *{1, 2, 12.1} °{13.4, 4} = {1.13.4, 1.4, 2.13.4, 2.4, 12.1.13.4, 12.1.4}*

Corollary 3.1: $|J| = W(A)$

Proof:

Follows directly from the fact that J is the set of all $f_i(n) \in A$, such that $a_i \neq 0$.

For example if A= 1001, W(A) = 2, J = {1, 4} and W(A) = |J|.

Definition 3.5: For a generator sequence $A = f_1(n).f_2(n)...f_k(n)$, a seed set associated with A is a set of base-r numerals defined over J as follows:

$$A_j = \{f_1(n).f_2(n)...f_{j-1}(n).0, f_1(n).f_2(n)...f_{j-1}(n).1,..., f_1(n).f_2(n)...f_{j-1}(n)(f_j(n)-1) : j \in J\}$$

$\forall\, 1 \leq j \leq k$.

Example 3.11: The seed sets of an n-cube generator $A=2$ are: $A_1 = \{0, 1\}$.

Example 3.12: Let $A=235$ be the generator for a LRN. The seed sets associated with A are: $A_1 = \{0, 1\}$, $A_2 = \{2, 0, 21, 22\}$, and $A_3 = \{230, 231, 232, 233, 234\}$.

Example 3.13: $A = n$ is the generator for the n-star graph. The seed sets associated with A are: $A_1 = \{0, 1, \ldots, n-1\}$.

We are now ready to formally define the vertex set (node set) of a LRN. We won't yet give a complete definition of G_n, the graph of a LRN, because the way the size of a LRN grows is independent from the way the nodes in a LRN are interconnected. Therefore, the edge set (links) will solely rely on a well-defined interconnection specification, while the size of the node relies solely on a linear recurrence equation. Since the Graph of a LRN depends on the size of the network, that is in turn influenced by the generator of the LRN, we will write G_n (A) to denote the graph of a LRN whose number of nodes depends on A.

Definition 3.6: Let A be the generator of a LRN. G_n (A) = (V_n(A), E_n(A)) denotes the graph of the LRN. V_n(A) is the set of vertices (nodes) of the graph and is defined as follows:

$$V_0(A) = \phi$$
$$V_n(A) = \bigvee_{j \in J} A_j \, o V_{n-j}(A)$$
$$\text{Note that } A_j V_{n-j}(A) = \phi, \forall n < j$$

The above definition describes precisely how to generate the labels of a LRN. The method is well defined and it satisfies the recurrence equation described by A.

Example 3.14: $A = 2$ is the generator of the n-cube since the n-cube's size can be described by the recurrence relation $X_n = 2\ X_{n-1}$. Let's apply the above definition to generate the node set of the 3-cube. $J = \{1\}$ and the seed sets are $A_1 = \{0, 1\}$.

$$
\begin{aligned}
V_0(A) &= \varnothing \\
V_1(A) &= A_1 V_0(A) = \{0,1\} \circ \varnothing = \{0,1\} \\
V_2(A) &= A_1 V_1(A) = \{0,1\} \circ \{0,1\} \\
&= \{00,01,10,11\} \\
V_3(A) &= A_1 V_2(A) = \{0,1\} \circ \{00,01,10,11\} \\
&= \{000,001,010,011,100,101,110,111\}
\end{aligned}
$$

As we would expect, there are $2^3 = 8$ nodes in a 3-cube where the labels are

{000,001,010,011,100,101,110,111}

Example 3.15: Let $A = n \rightarrow J = \{1\}$ and the seed sets are $A_1 = \{0, \ldots, n-1\}$. Let's generate the node set of the LRN generated by $A = n$ for n = 3. Note that $V_n(A) = A_1\ V_{n-1}(A)$.

$$
\begin{aligned}
V_0(A) &= \varnothing \\
V_1(A) &= A_1 V_0(A) = \{0\} \circ \varnothing = \{0\} \\
V_2(A) &= A_1 V_1(A) = \{0,1\} \circ \{0\} \\
&= \{00,10\} \\
V_3(A) &= A_1 V_2(A) = \{0,1,2\} \circ \{00,10\} \\
&= \{000,010,100,110,200,210\}
\end{aligned}
$$

As we would expect, the solution to the recurrence equation $X_n = n\ X_{n-1}$ is n!. Therefore we should obtain 3! = 6 elements and they are:

{000,010,100,110,200,210}

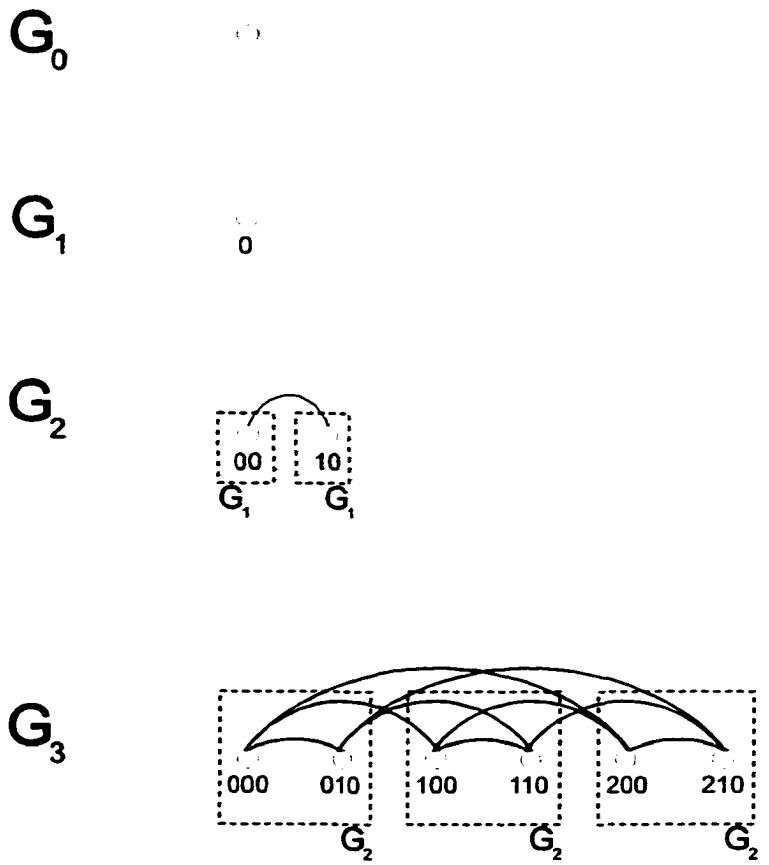


Figure 22: $G_n(A)$ for A = n and n = 3.

Figure 22 shows $G_3(A)$ if interconnection was dictated by hamming distance. Note that in $G_3(A)$ there are 3 copies of $G_2(A)$.

Example 3.16: Let $A = 2.1$. $J = \{1,2\} \rightarrow A_1 = \{0, 1\}$ *and* $A_2 = \{20\}$. Let's construct the node set of the LRN generated by $A = 21$.Let's generate the node set $V_3(A)$. Note that $V_n(A) = A_1 V_{n-1}(A) \cup A_2 V_{n-2}(A)$.

$$V_0(A) = \varnothing$$

$$V_1(A) = A_1 V_0(A) \cup A_2 V_1(A)$$
$$= \{0,1\} \circ \varnothing \cup \varnothing = \{0,1\}$$

$$V_2(A) = A_1 V_1(A) \cup A_2 V_0(A)$$
$$= \{0,1\} \circ \{0,1\} \cup \{20\} \circ \varnothing$$
$$= \{00,01,10,11,20\}$$

$$V_3(A) = A_1 V_2(A) \cup A_2 V_1(A)$$
$$= \{0,1\} \circ \{00,01,10,11,20\} \cup \{20\} \circ \{0,1\}$$
$$= \{000,001,010,011,020,100,101,110,111,120,200,201\}$$

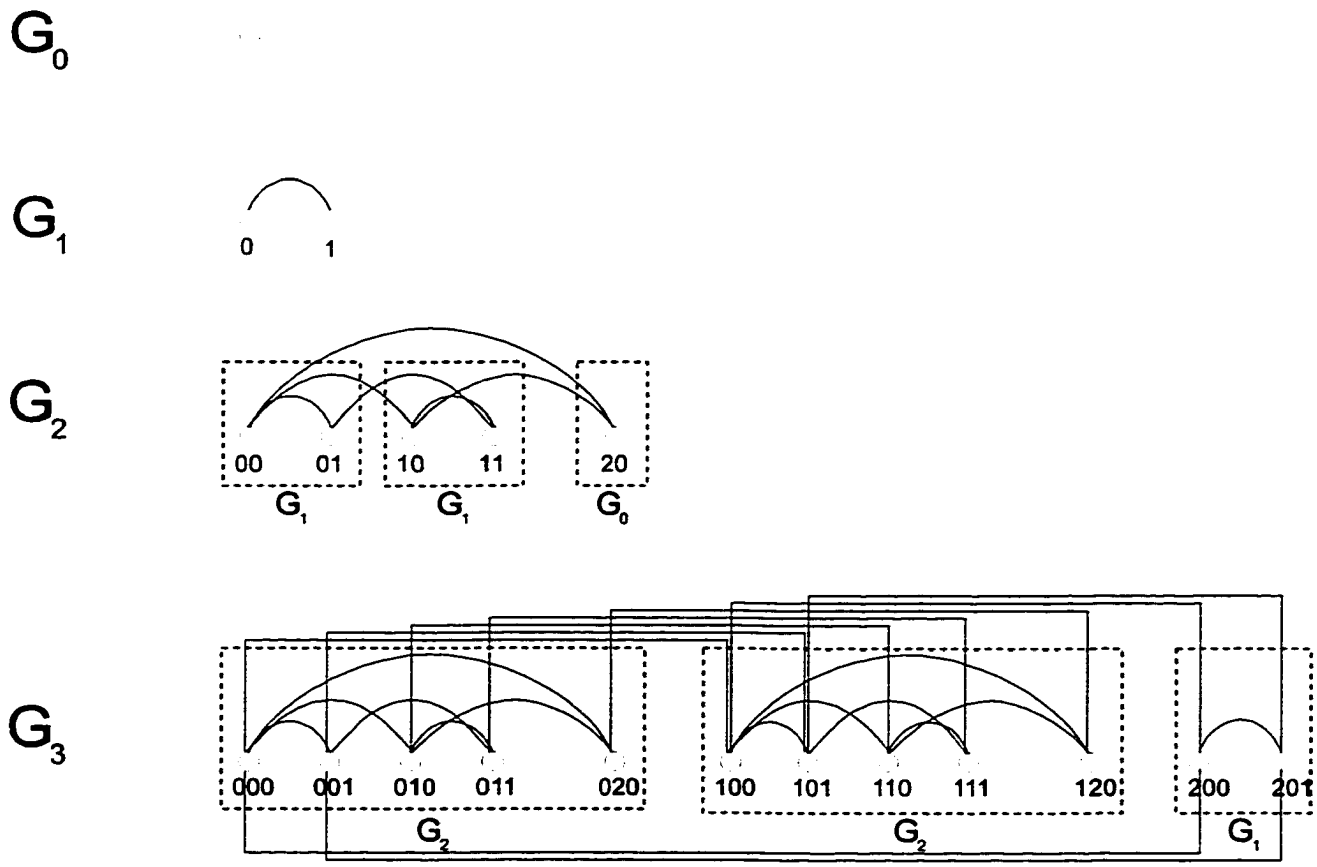


Figure 23: $G_n(A)$ for A = 2.1 and n = 3.

As we would expect, there are 12 nodes in a LRN whose size is modeled by the recurrence equation: $X_n = 2\,X_{n-1} + X_{n-2}$ for n = 3. To see how we get this number, assume initial condition $X_0 = 1$, and work the numbers up. $X_0 = 1$, $X_1 = 2$, $X_2 = 5$, and $X_3 = 12$.

Figure 23 shows $G_3(A)$ if interconnection was dictated by hamming distance. Note that there are 2 copies of $G_{n-2}(A)$ and one copy of $G_{n-1}(A)$ in $G_3(A)$.

Next, we will describe several static properties of LRNs. These properties are related to the node set of the network. But first we will lay out some more properties of the seed sets, which will be used in proving that the node labels generated by the above process are unique, which is a necessary condition, since no two processing elements can have the same physical labels.

Corollary 3.2: The number of distinct seed sets of a generator is $W(A)$.

Proof: to each $f_j(n)$ where $j \in J$, corresponds a seed set A_j. Therefore the number of seed sets is equal to the number of non-zero elements of $A \rightarrow |J| = W(A)$.

Example 3.17: Let $A=1001$. Then $Wt(A) = 2$. Thus $J = \{1, 4\}$. Therefore the two seed sets A_1 and A_4 of A are: $A_1 = \{0\}$, and $A_4 = \{1000\}$. The number of seed sets is exactly equal to $W(A)$.

Example 3.18: Let $A=1101011$. Then $W(A) = 4$. Thus $J = \{1, 2, 4, 6, 7\}$. Therefore the four seed sets A_1, A_2, A_4, A_6 and A_7 are: $A_1 = \{0\}$, $A_2 = \{10\}$, $A_4 = \{1100\}$, $A_6 = \{110100\}$ and $\{A_7 = \{1101010\}$. $W(A) = |J|$.

Example 3.19: Let $A=20305$ be a generator. Then, $W(A) = 3$ and $J = \{1, 3, 5\}$. The seed sets of A are: $A_1 = \{0, 1\}$, $A_3 = \{200,, 201, 202\}$, and $A_3 = \{20300, 20301, 20302, 20303, 20304\}$. $W(A) = |J|$.

Corollary 3.3: $s \in A_j \rightarrow |s| = j$ where A_j is a seed set of A.

Proof:

It follows directly from the way seed sets are constructed.

If $s \in A_1 = \{0, 1 \ldots f_1(n)-1\}$ then $|s| = 1 = j$ and for all $2 \leq j \leq k$, if $s \in A_j = (f_1(n). f_2(n) \ldots f_{j-1}(n)) \circ \{0, 2\ldots, (f_j(n)-1)\} = \{f_1(n) \ldots f_{j-1}(n)0, \ldots, f_1(n)\ldots f_{j-1}(n) (f_j(n)-1)\}$ then $|s| = |f_1(n).f_2(n)\ldots f_{j-1}(n)| + 1 = j - 1 + 1 = j$.

Corollary 3.4: s_1 and $s_2 \in A_j \rightarrow s_1 \neq s_2$ where A_i and A_j are seed sets of A. The labels in a seed set are distinct.

Proof:

If s_1 and $s_2 \in A_1 = \{0, 1 \ldots f_1(n)-1\}$, then obviously $s_1 \neq s_2$ since each s_i corresponds to a different element in A_1 and no two elements in A_1 are identical since they were

constructed by starting at 0 and incrementing each element by 1 until $f_1(n)-1$ was reached. On the other hand if s_1 and $s_2 \in A_j = (f_1(n). f_2(n) \ldots f_{j-1}(n)) \circ \{0, 2, \ldots, (f_j(n)-1)\} = \{f_1(n) \ldots f_{j-1}(n)0, \ldots, f_1(n) \ldots f_{j-1}(n) (f_j(n)-1)\}$ then also $s_1 \neq s_2$ since s_1 and s_2 differ in the jth base-r numeral. s'_1 and $s'_2 \in A_j \rightarrow s'_1$ and $s'_2 \in \{f_1(n) \ldots f_{j-1}(n)0, \ldots, f_1(n) \ldots f_{j-1}(n) (f_j(n)-1)\} = (f_1(n). f_2(n) \ldots f_{j-1}(n)) \circ \{0, 2 \ldots (f_j(n)-1)\} \rightarrow s'_1 = s \circ x$ and $s'_2 = s \circ y$ where x and $y \in \{0, 2 \ldots (f_j(n)-1)\}$.

Since x and $y \in \{0, 2 \ldots (f_j(n)-1)\}$, *and* $x \neq y$ therefore $(s'_1 = s \circ x) \neq (s'_2 = s \circ y)$.

Corollary 3.5: $s_1 \in A_i$ and $s_2 \in A_j \rightarrow s_1 \neq s_2$ where A_i and A_j are seed sets of generator A.

Proof:

If $i = j$ then s_1 and $s_2 \in A_i$ and by corollary 3.4, $s_1 \neq s_2$.

Otherwise $i \neq j$ and $s_1 \in A_1 \rightarrow |s_1| = i$, $s_2 \in A_j \rightarrow |s_2| = j \rightarrow |s_1| \neq |s_2| \rightarrow s_1 \neq s_2$.

If two strings are equal they must have the number of symbols and all the corresponding symbols have to be identical.

Corollary 3.6: $v \in A_i \circ V_{n-i}(A) \rightarrow |v| = n$ where A_i is a seed set of a generator A.

Proof:

If $n < j$, $A_i \circ V_{n-i}(A) = \varnothing$. Since there are no s in $\varnothing$, this case does not apply.

If $n = j$, $V_0(A) = \varnothing$ and therefore $v \in A_n$. By corollary 3.3, $|v| = n$.

Otherwise, $j < n$ and thus the label of v is made up of the concatenation of an element of A_j, whose length is j, with an element of V_{n-j} whose length is $n-j$: $v \in \{f_1(n) \ldots f_{j-1}(n)0, \ldots, f_1(n) \ldots f_{j-1}(n)\,(f_j(n)-1)\} \rightarrow |v| = j + (n - j) = n$.

Corollary 3.7: $A_i \circ V_{n-i}(A)$ and $A_j \circ V_{n-j}(A)$ are disjoint sets for all $i \neq j$. That is $A_i \circ V_{n-i}(A) \cap A_j \circ V_{n-j}(A) = \varnothing$.

Proof:

Suppose that $s \in A_i \circ V_{n-i}(A) \cap A_j \circ V_{n-j}(A)$ and $i < j$.

$A_i = \{f_1(n).f_2(n) \ldots f_{i-1}(n).0, f_1(n).f_2(n) \ldots f_{i-1}(n).1, \ldots, f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n)-1\}$

$A_j = \{f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n) \ldots f_{j-1}(n).0, f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n) \ldots f_{j-1}(n).1, \ldots f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n) \ldots f_{j-1}(n).f_j(n)-1\}$

Since $s \in A_i \circ V_{n-i}(A) \cap A_j \circ V_{n-j}(A)$ then $s \in A_i \circ V_{n-i}(A)$ and $s \in A_j \circ V_{n-j}(A)$

$s \in A_i \circ V_{n-i}(A) \rightarrow s \in \{f_1(n).f_2(n) \ldots f_{i-1}(n).0, f_1(n).f_2(n) \ldots f_{i-1}(n).1, \ldots, f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n)-1\} \circ V_{n-i}(A)$

$s \in A_j \circ V_{n-j}(A) \rightarrow s \in \{f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n) \ldots f_{j-1}(n).0, f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n) \ldots f_{j-1}(n).1, \ldots f_1(n).f_2(n) \ldots f_{i-1}(n).f_i(n) \ldots f_{j-1}(n).f_j(n)-1\} \circ V_{n-j}(A)$.

Let's examine s at the ith position:

$s \in A_i \rightarrow s(i) \in \{0, 1 \ldots a_i-1\}$ and $s \in A_j \rightarrow s(i) = a_i$. Contradiction.

$\rightarrow A_i \circ V_{n-i}(A) \cap A_j \circ V_{n-j}(A) = \varnothing$.

Corollary 3.8: Let $v \in V_n(A) \rightarrow |Label(v)| = n$.

Proof:

For $n = 0$, $V_n(A) = \varnothing \rightarrow$ the node set is empty and there fore there are no labels in $V_0(n)$ $\rightarrow \forall v \in V_0(A), |Label(v)| = 0 = n$.

Otherwise, $v \in V_n(A) = \bigcup_{j \in J} A_j \circ V_{n-j}(A)$. $\rightarrow v$ belongs to some $A_j \circ V_{n-j}(A)$.

By corollary 3.6, $|Label\ (v)| = n$.

Corollary 3.9: The vertices labels generated by LRM are unique in $V_n(A)$.

That is if $v_1, v_2 \in V_n(A)$ and $v_1 \neq v_2 \rightarrow Label(v_1) \neq Label(v_2)$.

Proof:

By corollary 3.7, the sets $A_j \circ V_{n-j}(A)$ are disjoint $\rightarrow \cap A_j \circ V_{n-j}(A) = \varnothing$ over all $j \in |J|$. So no two $A_j \circ V_{n-j}(A)$ produce the same label $\rightarrow$ if $Label\ (v_1) \in A_i \circ V_{n-i}(A)$ and $Label\ (v_2) \in A_j \circ V_{n-j}(A) \rightarrow Label\ (v_1) \neq Label\ (v_2)$. Also by corollary 3.4, no two elements of A_j are the same $\rightarrow s_1$ and $s_2 \in A_j \rightarrow s_1 \neq s_2$. It follows that s_1 and $s_2 \in A_j \circ V_{n-j}(A) \rightarrow s_1 \neq s_2$. Let $s_1 = Label\ (v_1)$, $s_2 = Label\ (v_2)$ we get $Label\ (v_1)$ and $Label\ (v_2) \in A_j \circ V_{n-j}(A) \rightarrow Label\ (v_1) \neq Label\ (v_2)$.

We have shown that the process described is a recipe to generate the node set of a LRN. The node labels that are generated are unique and their length is the same as prescribed by the recurrence equation used to model how the network scales. It is also essential to know an upper bound of the number of nodes N in a LRN whose size is modeled by $X_n = f_1(n)X_{n-1} + f_2(n)X_{n-2} + ... + f_k(n)X_{n-k}$.

Problem statement: Given an base-r LRN characterized by the linear recurrence equation $X_n = f_1(n)X_{n-1} + f_2(n)X_{n-2} + ... + f_k(n)X_{n-k}$, what is the upper bound of the number of the nodes N in X_n?

$$
\begin{aligned}
X_n &= f_1(n)X_{n-1} + f_2(n)X_{n-2} + ... + f_k(n)X_{n-k} \\
&\le f_1(n)X_{n-1} + f_2(n)X_{n-1} + ... + f_k(n)X_{n-1} \\
&= (f_1(n) + f_2(n) + ... + f_k(n))X_{n-1} \\
&= cX_{n-1} \text{ where } c = f_1(n) + f_2(n) + ... + f_k(n) \\
&= c^n X_0 \\
&\Rightarrow X_n = N = O(c^n)
\end{aligned}
$$

The above analysis shows that the number of nodes in an base-r LRN X_n is bounded above by c^n where $c = (f_1(n) + ... + f_k(n))$. A tighter bound can be found but for now we will do with the above result.

Lemma 3.1: Let 0^n denote a string made of solely n zeros. $0^n \in V_n(A)$ for all A

Proof:

Let A be a generator so that $A = f_1(n) f_2(n) \ldots f_k(n)$.

$A_1 = \{0, \ldots, f_1(n) - 1\}$ so that the first time for $0 \in V_1(A)$.

Suppose that $0^n \in V_n(A)$, and we will prove that $0^{n+1} \in V_{n+1}(A) = \{0^n, \ldots\}$.

$V_{n+1}(A) = A_1 \circ V_n \cup \ldots = \{0 \ldots f_1(n) - 1\} \circ \{0^n, \ldots\} \cup \ldots = \{0\,0^n, \ldots\} = \{0^{n+1}, \ldots\}$

$\rightarrow 0^{n+1} \in V_{n+1}(A)$.

Therefore, for all A and n, $0^n \in V_n(A)$. □

Lemma 3.2 Let $v \in V_n(A)$: $v = v_1 v_2 \ldots v_n$ such that $v_k \neq 0$. Then $v = v_1 v_2 \ldots 0$ is also in $V_n(A)$.

Proof:

We will proceed by induction on the length of the label v.

Base case:

Let $n = |v| = 1 \rightarrow V_n(A) = V_1(A) = A_1 = \{0, \ldots, f_1(n)-1\}$

$v \in V_n(A) \rightarrow 0 \in V_n(A)$

Inductive step:

Assume that $v = v_1 v_2 \ldots v_n \in V_n(A) \rightarrow v_1 v_2 \ldots 0 \in V_n(A)$, then

$v' = v_0 v_1 v_2 \ldots v_n \in V_{n+1}(A)$ is an element of $A_1 \circ V_n(A)$ where $v_0 \in A_1$ and $v_1 v_2 \ldots v_n \in V_n(A)$ as stated by the induction hypothesis.

Since $v' \in V_{n+1}(A)$, then automatically follows that $v_0\, v_1v_2\ldots0 \in V_{n+1}(A)$ since $v_0 \in A_1$ and $v_1v_2\ldots0 \in V_n(A)$ as stated by the induction hypothesis.

Therefore, for all $v = v_1v_2\ldots v_n \in V_n(A)$, $v_1v_2\ldots0 \in V_n(A)$.

Lemma 3.3 For all $v = v_1v_2\ldots v_k\, 0^{n-k} \in V_n(A)$, $v_1v_2\ldots v_{k-1}\, 0^{n-k+1} \in V_n(A)$ where $v_k \neq 0$.

Proof:

$v = v_1v_2\ldots v_k\, 0^{n-k} \in V_n(A) \rightarrow v' = v_k\, 0^{n-k} \in V_{n-k+1}(A)$.At some earlier stage of the labeling process, $v_k\, 0^{n-k}$ must have existed in order for $v_1v_2\ldots v_k\, 0^{n-k}$ to exist.

If $v_k\, 0^{n-k}$ is a node label in $V_{n-k+1}(A)$, then so is $0^{n-k+1} \rightarrow 0^{n-k+1} \in V_{n-k+1}(A)$

Since $v_k\, 0^{n-k}$ and 0^{n-k+1} are node labels at an early stage of the labeling process and $v = v_1v_2\ldots v_k\, 0^{n-k}$ was created by that process, then so will $v_1v_2\ldots v_{k=1}\, 0^{n-k+1}$. $v_1v_2\ldots v_k\, 0^{n-k}$ is obtained by concatenating $v_1v_2\ldots v_{k=1}$ with $v_k\, 0^{n=k}$ and $v_1v_2\ldots v_{k=1}\, 0^{n-k+1}$ was obtained by concatenating $v_1v_2\ldots v_{k=1}$ with 0^{n-k+1}. □

1.1 Decomposition of a node label

We will present in this subsection a way to decompose the label of a node. This algorithm will be used to develop routing and embedding strategies, and it will also be used to demonstrate several properties of LRNs.

A node label is constructed recursively from elements of A_j where A_j is a seed set of A. The composition of a node label will reflect all different seed sets that are used to

construct the label. This decomposition is a one way function, from the node label to a sequence of A_j, since it does not actually give us the components that constitute the label, but rather a sequence of the sets to which those components belonged.

Definition 3.7: Let $A = f_1(n).f_2(n)...f_k(n)$ and G_n (A) be a generator and a graph of a LRN respectively. For all nodes v in $V_n(A)$, v can be decomposed as sequence of base-r numerals from A_j, where A_j are seed sets of A. The decomposition of the label, DEC is a one way function from the node label to a sequence of A_j. DEC: $V_n(A) \rightarrow \{A_j\} j \in J\}$.

Example: Let $A = 23. \rightarrow A_1 = \{0, 1\}$ *and* $A_2 = \{20. 21, 22\} \rightarrow V_3(A) = \{0.0.0, 0.0.1, 0.1.0, 0.1.1, 0.2.0, 0.2.1, 0.2.2, 1.0.0, 1.0.1, 1.1.0, 1.1.1, 1.2.0, 1.2.1, 1.2.2, 2.0.0, 2.0.1, 2.1.0, 2.1.1, 2.2.0, 2.2.1\}$.

1.0.1 can be decomposed as follows: $A_1\ A_1\ A_1$

2.1.0 can be decomposed as follows: $A_2\ A_1$

Decomposition algorithm

Next, we will propose a recursive technique to decompose the node labels of an LRN. To build the labels, we recursively appended, from the left, members of A_j repeatedly. Now, to construct the decomposition, we will parse the labels from left to right, and decide from what A_j the base-r numeral came. The process will be repeat for

every base-r numeral, or group of base-r numeral. We can easily distinguish to what A_j a group of base-r numerals belong, since each A_j has unique prefix.

```
DEC(v) //v is a label of a node
i = 1
s = empty
decomposition = empty
for(int i = n-1; i < n; i++)
        s = s + v_i
        if s ∈ A_j for all j ∈ J
                decomposition = decomposition + A_j
                s = empty
return decomposition
```

Complexity analysis of the decomposition algorithm

The decomposition algorithm steps through each base-r numeral, v_i, of the node label from left to right to verify if it is a valid seed set. If that is the case, that A_j is appended to the decomposition. Otherwise, the next base-r numeral is appended to the previous one, and the loops starts all over. The for loop is executed exactly n times. Each iteration does one constant operation ($s = s + v_i$), and a look up (if $s \in A_j$ for all $j \in J$).

The look up actually need to check if s in any A_j. There are k A_j. There for the look up takes k steps. If the lookup if successful, two constant operations are performed. The runtime of the decomposition algorithm can be expressed as follows:

$$\begin{aligned} T(n) &= n(1+k) \\ &= n+nk \\ &\leq nk+nk \\ &\leq 2nk \\ \rightarrow T(n) &= O(nk) \end{aligned}$$

Further analysis of either static properties or dynamic properties of LRNs requires knowledge of the edge which so far we have not yet well-defined except the fact that it is the set which define the links between the nodes of the system.

The edge set of the network can be defined in several ways to satisfy certain requirements such as node degree, parallel paths and diameter. A point that needs to be emphasized in this thesis is that the way the nodes are interconnected is independent of the number of node in the system. This relaxes certain restriction otherwise imposed on the systems to be designed. By this we mean that, since the interconnection is independent of the number of the nodes in a system, the analysis of similarly interconnected systems will be general. Algorithms generated for a system with X numbers of nodes should be applicable to a system with Y nodes as long as they define interconnection similarly.

For example, we will define a class of LRNs that are interconnected by means of hamming distance. That is two nodes will be connected if and only if their hamming

distance is equal to some constant number. We will generate routing algorithm for that class and those routing algorithms should be applicable to all member of that class. We should mention thought that those algorithms might not be optimal for a specific member of the class. In many occasions, network designers utilize certain nice properties about a topology to optimize algorithms used on that topology.

2 The class of LRN

The class of LRN is a very large class of interconnection networks. All these interconnection networks share the property that their node sets are modeled using some well-defined linear recurrence equation (More properties will be are shared as we advance in this study). The node set is accordingly generated. The edge set depends on the application the interconnection network is designed for, and other static and dynamic properties it has to respect. Figure 24 depicts the hierarchy of LRNs. It is rather clear that the root of the tree is made of all LRNs. At this level, no information is yet known about the edge set. As we go down the hierarchy, we get further specialized networks. Networks are classified by their interconnection. That is, if two networks are interconnected using the same kind of interconnection function, those two networks belong to the same class.

LRNs at the same level share the same routing and embedding algorithms. By this we mean that if LRN_1 and LRN_2 belong to the same level (same class of interconnection), Algorithms defined at that level are applicable to LRN_1 and LRN_2, and further more all other properties, static or dynamic, are applicable at that level. As an example all the

LRN share the property that their node set is bounded above by $O(c^n)$. Another example is that all LRN share static routing algorithms that require global knowledge of the

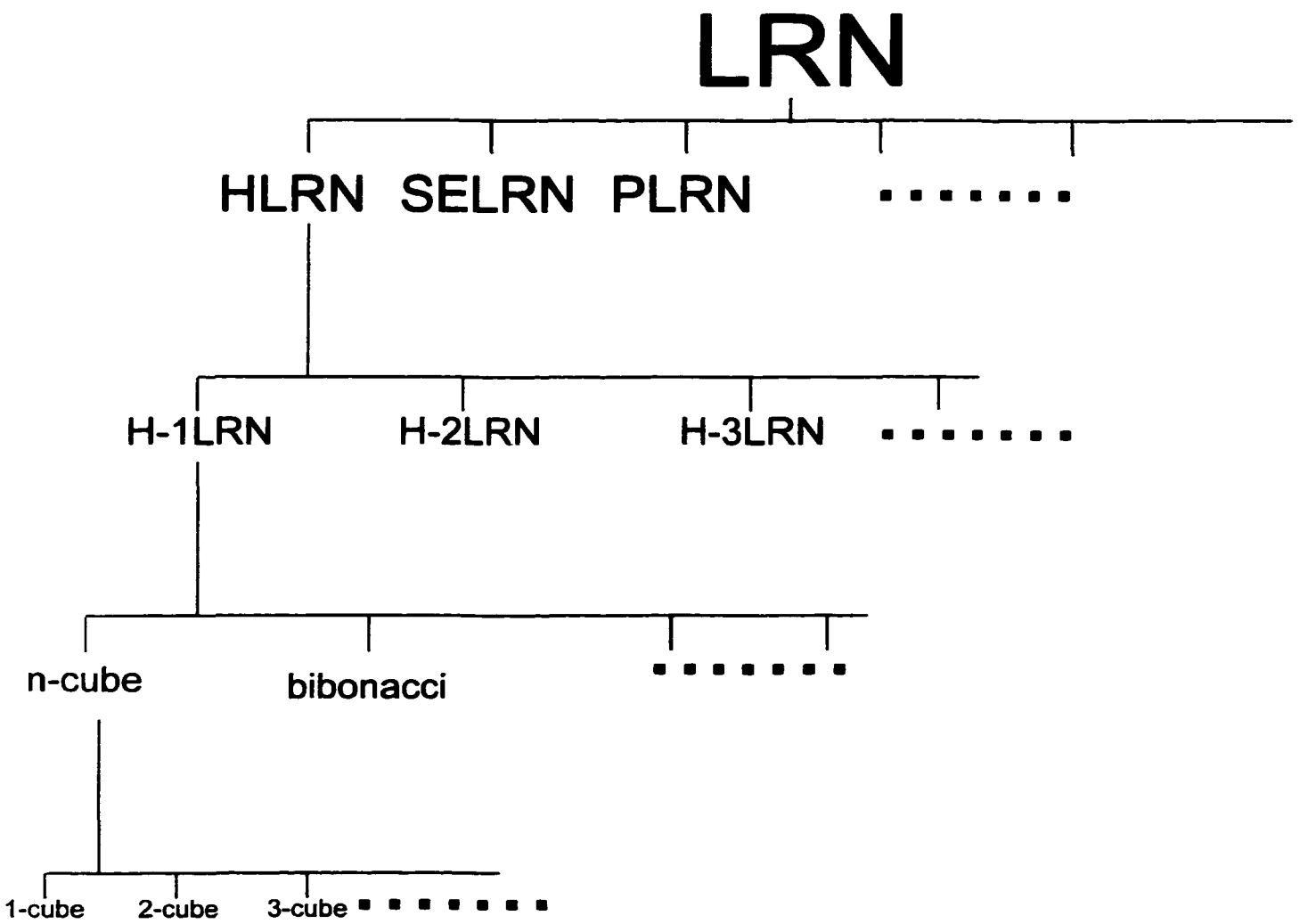


Figure 24: The LRN Hierarchy.

network, such as Dijkstra's algorithm. An advantage of this classification is that it is sometimes difficult to study properties a certain network, and/or it might be difficult to come up with routing strategies for that network. In that case it is simply a matter of

identifying what layer the LRN belongs, and that permits the use of existing algorithms already defined for a different LRN that belongs to the same level.

The Top most Level, labeled as LRN, contains all networks whose sizes can be described by a linear recurrence equation, that is all LRNs. At this level, the edge set is note specified yet. In the second level we specify the kind of interconnection. At this level, LRNs subdivided to several sub-classes depending on the method used to interconnect the node of the LRN. For example, all the LRNs, which will interconnect their nodes given a hamming distance, will belong to the same sub-class, and that sub-class will be named HLRN. HLRN stands for Hamming LRN. Notable members of HLRN are the n-cube and the Fibonacci networks.

There are many ways to interconnect nodes, and every one of these methods result in a new class of networks which is defined at the second level of this hierarchy. Some of the above-mentioned sub-classes are SELRN and PRLN. SELRN stands for Shuffle-Exchange LRN. Notable members of this class are the shuffle-exchange network, and the Omega network. PLRN stands for Permutation LRN and a notable member of this class is the n-star.

We need to mention that there are two distinct kinds of linear recurrence equations, namely: constant and non-constant coefficient linear recurrence equations. By constant coefficient linear recurrence, we mean those linear recurrences with all coefficients constant.

An example of constant coefficient linear recurrence is $X_n = 2\,X_{n-1} + 3\,X_{n-2}$. And by non-constant coefficient linear recurrence, we mean those linear recurrences with at least one non-constant coefficient. An example of non-constant coefficient linear recurrence is X_n

$= X_{n-1} + n X_{n-2}$. Since each LRN is associated to a linear recurrence of the form $X_n = f_1(n)X_{n-1} + f_2(n)X_{n-2} + ... + f_k(n)X_{n-k}$, and therefore has a generator $A = f_1(n).f_2(n)...f_k(n)$, a LRN is said to be constant or to have a constant generator if $\forall$ n and $\forall i$: $1 \leq i \leq k, f_i(n) = a_i$, where a_is are constants. So we say that $A = a_1.a_2...a_k$ is the generator of a constant LRN. Otherwise, a LRN is said to be non-constant or to have a non-constant generator if there exists at least one i: $1 \leq i \leq k$, and $f_i(n)$ is not constant.

The notion of constant and non-constant generator has an impact on the complexity of the computations involved in setting up the node set and other structures since for every different n, $f_i(n)$ is different, but it has no effect on the hierarchy and the previous discussion is valid for either types, since whether constant or not, once the nodes are generated, the interconnection function will be applied to the labels of the nodes regardless.

Going down to the next level of the hierarchy, we have members of specific interconnection classes. For example following the tree spawned by HLRN, we find two new sub-classes that are the n-cube and the Fibonacci networks. Theses are only two of the many members of the class of HLRN. Members of this class share all properties and algorithms defined at the level of HLRN, but as mentioned before, a specific member of the class might have certain algorithms defined only for it and not applicable for other members of the same class. To well understand this, here is a concrete example related to the class of HLRN.

As we mentioned earlier, both the n-cube and the Fibonacci networks are HLRNs. A general static routing algorithm can be used to route messages in either network. But this

algorithm is expensive since it requires storage. Thus if the size of the network grows, the table has to be reconstructed. The n-cube has an optimal routing function as described in this chapter. This routing function is not static and requires no storage, and is therefore very efficient. But this routing function is not applicable to the Fibonacci networks, even if both the n-cube and the Fibonacci networks exist on the same level of the hierarchy. If the n-cube's routing functions are applied to the Fibonacci network, errors will occur since the Fibonacci network is incomplete. The optimal routing function defined for the n-cube takes advantage not only of the interconnection function of the network, but also of other topology properties particular to the n-cube.

3 Approaches to selecting the edge set for a LRN.

The LRM defines the node set only. The edge set has to be defined independently to fit certain constraints such as cost, performance, diameter, and applications for which the LRN is to be optimized.

The cost of the network is a function of the number of processing elements, which constitute the node set, and the number of links, which constitute the edge set. We can therefore subdivide the cost of a LRN into two parts: the cost of the node set and the cost of the edge set. The generator of the LRN determines the cost of the node set. When the network designer chooses the Linear recurrence to associate with the size of the network, the cost of the node set is well defined for any dimension n of that network.

But the edge set is determined at a later stage of the design. The designer might have a budget to respect, and therefore he is constrained to a maximum amount of links,

which he then has to use to interconnect the nodes and provide the best possible/efficient communication.

The network designer might be asked to interconnect the nodes given a well-defined function, in which case he has no control over the performance of the system, neither does he have control over the cost of the network.

The edge set has to be defined such that the network provides a minimum acceptable performance. An example is when a network designer is putting together a network for real-time parallel applications, which requires very efficient communication, so that response time is almost immediate, and delay is minimized.

Certain network designers seek to reduce the diameter of existing LRN. For example, the diameter of the Hypercube is n. To reduce the diameter of the cube, one has to introduce additional links, which will reduce the diameter of the network, but at the same time, the cost of the network increases. Adding additional links does not only reduce the diameter of a network, but also give rise to more reliable communication. If one link is faulty, an alternate link can be used to establish communication.

It is usually the case that a network designer has to satisfy several of the above mentioned constraints and more. It remains that the most important task is to optimize communication, minimize delay, and provide reliable communication. Designers seek to build relatively cheap network and reduce the delay of communication, in which one of the most important factors is the diameter of the network. Mathematical models are often used to reduce the diameter of network and therefore obtain variations of existing networks but with lower diameter. One way this can be accomplished is by introducing additional links into the network as mentioned before. This method brings about an

explosion in the cost. Most designers seek to maintain the same cost while reducing the diameter and therefore faster communication. One way to do this is to use redundant set of basis. Some networks such as the Hypercube are modeled using a non-redundant basis. For example a 3 dimensional Hypercube can be seen as a 3-dimensinal vector space and therefore spanned by the basis (0 0 1), (0 1 0) and (1 0 0). By introducing a redundant basis such as (0 1 1), (1 1 0) and (1 0 1), the diameter of the Hypercube can be reduced from n to approximately n/2. More about redundant basis and their role in reducing diameter will come in a later chapter.

In the remainder of the thesis will examine (a) several LRN families, (b) develop general-purpose algorithms that can be applied to a wide variety of LRNs, and (c) examine specific LRN topologies and develop optimal algorithms for them, and show some of their applications.

We will also seek to study the static and dynamic properties of certain LRNs and show how this methodology can be utilized in studying the members of the large class of Linear Recursive Networks.

4 HLRN (Hamming distance LRN)

The class of HLRN is of special interest. A large number of well-studied interconnection networks connect their nodes based on the hamming distance property. The hamming distance of two nodes, v_1 and v_2, is the number of base-r numerals that differ in their labels. A LRN is also a HLRN if the nodes in that LRN are interconnected

based on some function of the hamming distance. Usually, two nodes are interconnected if their hamming distance is one.

We will study the properties of HLRN, which will be used in subsequent chapters. We will prove the most fundamental property of any interconnection networks: Connectivity. We will prove other properties that will help us generate routing and embedding algorithms.

Definition 3.8: Let $G_n(A)$ be a HLRN for some generator A. Let v_1 and $v_2 \in V_n(A)$. The edge $(v_1, v_2) \in E_n(A)$ if and only if $H(v_1, v_2) = 1$. i.e. the vertices v_1 and v_2 conflict in one base-r numeral.

4.1 Connectivity

The graph that underlies an interconnection network must be a connected graph. Otherwise, some communication will not be available between al the nodes in the system. The graph, $G_n(A)$, of an HLRN is a connected graph.

Lemma 3.4: The graph $G_n = (V_n, E_n)$ of a HLRN is a connected graph. Where V_n is the vertex set of G_n: $V_n(A) = \cup A_j \circ V_{n-j}(A)$.

This lemma will be proved in two parts. Part1 will prove that each individual $A_j \circ V_{n-j}(A)$ induce a connected graph. Part2 will show that all different $A_j \circ V_{n-j}(A)$ are connected and therefore the resulting graph, G_n is connected.

Proof Part1: By strong induction on n: n is the number of symbols used to label the nodes of the HLRN.

Base case:

For n = 0, $| V_0 (A) | = 0$, therefore the graph of the HLRN is empty and we can then say that all the nodes in the corresponding HLRN are interconnected.

For n = 1, $| V_1 (A) | \geq 0$, therefore $| V_1 (A) |$ could be equal to 0: this case is similar to when n = 0. $| V_1 (A) |$ could be equal to 1 in which case the graph of the HLRN is made up of one node and is obviously interconnected. Otherwise $| V_1 (A) | \geq 2$, in which case the labels of the HLRN are the elements of A_1 and therefore their length is 1. Since the length of the labels of any node in the HLRN is 1, the Hamming distance between them is at most one, and therefore, they are all interconnected. The hamming distance could not be 0 since all the elements of any A_I are distinct are proved in the previous chapter. Thus the hamming distance between all the elements of A_I is exactly 1. Thus, the graph of the HLRN is a connected graph.

Recursive step

We assume that all HLRN G_j are connected graphs for $j < n$. We will show that G_n is also a connected graph. Meaning that if there exists a node v_1 in $V_n(A)$, then there exists a node $v_2$in $V_n(A)$ such that $H(v_1, v_2) = 1$

Let $v_1 \in V_n(A) \rightarrow v_1 \in \cup A_j \circ V_{n-j}(A)$.

$j < n \rightarrow G_{n-j}(A)$ is a connected graph $\rightarrow$ all the nodes in $V_{n-j}(A)$ are connected

$\rightarrow$ There exists $v_2 \in V_{n-j}(A)$ such that v_1 is connected to $v_2 \rightarrow H(v_1, v_2) = 1$

When we will combine A_j with $V_{n-j}(A)$ we will obtain new labels that are the concatenation of the elements of A_j and $V_{n-j}(A)$. (1)

Let $A_j = \{A_{j,1}, A_{j,2}, \ldots, A_{j,k}\}$ and let $V_{n-j}(A) = \{v_{n-j,1}\ v_{n-j,2}, \ldots, v_{n-j,m}\}$. We will obtain m * k new labels divided into k groups as follows:

$Group_1 = A_{j,1} \circ V_{n-j}(A) = A_{j,1} \circ \{v_{n-j,1}\ v_{n-j,2}, \ldots, v_{n-j,m}\} = \{A_{j,1}v_{n-j,1}\ A_{j,1}v_{n-j,2}, \ldots, A_{j,1}v_{n-j,m}\}$

$Group_2 = A_{j,2} \circ V_{n-j}(A) = A_{j,2} \circ \{v_{n-j,1}\ v_{n-j,2}, \ldots, v_{n-j,m}\} = \{A_{j,2}v_{n-j,1}\ A_{j,1}v_{n-j,2}, \ldots, A_{j,2}v_{n-j,m}\}$

.

$Group_k = A_{j,k} \circ V_{n-j}(A) = A_{j,k} \circ \{v_{n-j,1}\ v_{n-j,2}, \ldots, v_{n-j,m}\} = \{A_{j,k}v_{n-j,1}\ A_{j,k}v_{n-j,2}, \ldots, A_{j,k}v_{n-j,m}\}$

Each $Group_l$ is made up of elements of $V_{n-j}(A)$ concatenated with the same symbol. Since $H(v_1, v_2)$ = for all elements of $V_{n-j}(A)$, $H(s.v_1, s.v_2) = 1$ and therefore each group constitutes a connected graph.

Also all the different groups are connected. Each element of A_j differs by one single entry so that $H(A_{j,m}, A_{j,n}) = 1$.

The same elements from $V_{n-j}(A)$ is concatenated k times with the k different elements of A_j so that all theses new labels only differ in one entry leading to nodes that are connected. $H(A_{j,m}, A_{j,n}) = 1 \rightarrow H(A_{j,m}.v_{n-j,l}, A_{j,n}.v_{n-j,l}) = 1$. (2)

By combining (1) and (2), we can easily see that the graph induced from each A_j $V_{n-j}(A)$, G_n, is a connected graph. □

We have only showed connectivity of elements of A_j $V_{n-j}(A)$ internally so that graphs induced individually by A_j $V_{n-j}(A)$ are connected graphs. We also need to also show that there exists always at least one arc from a node of A_j $V_{n-j}(A)$ to a node of A_k $V_{n-k}(A)$ for

some j and k such that $1 \leq j \leq n$, and $1 \leq k \leq n$. Once this is shown the proof of connectivity is complete and, we could then say that there exists a path between any two nodes in $V_n(A)$.

We will demonstrate connectivity of the different $A_j\ V_{n-j}(A)$ by showing that there exists always at least two nodes, node $v_1 \in A_j\ V_{n-j}(A)$ and node $v_2 \in A_{j+1}\ V_{n-(j+1)}(A)$ so that there is a path form v_1 to v_2. Proving this is sufficient, since above can be translated to $A_1\ V_{n+1}(A)$ is connected to $A_2\ V_{n+2}(A)$, $A_2\ V_{n+2}(A)$ is connected to $A_3\ V_{n+3}(A)$, and by transitivity, $A_1\ V_{n+1}(A)$ is connected to $A_3\ V_{n+3}(A)$. The same process will apply in general to deduce that all $A_j\ V_{n-j}(A)$ are connected, and finally by combining this result with the connectivity of the graphs induced by $A_j\ V_{n-j}(A)$, we can conclude that the graph induced by $V_n(A)$, $G_n(A)$, is a connected graph.

Proof Part2:

Let j be any random integer such that $1 \leq j \leq n$. The graph induced by $A_j\ V_{n-j}(A)$ is connected to the graph induced by $A_{j+1}\ V_{n-(j+1)}(A)$ means that there exists two vertices $v_1 \in A_j\ V_{n-j}(A)$ and $v_2 \in A_{j+1}\ V_{n-(j+1)}(A)$ such that $H(v_1, v_2) = 1$.

Let $V_{n-(j+1)}(A) = \{l_1, l_2, \ldots\}$

$\rightarrow V_{n-j}(A) = A_1\ V_{n-j-1}(A) \cup \ldots = A_1\ V_{n-(j+1)}(A) \cup \ldots = \{A_1\ l_1, A_2\ l_2, \ldots\} \cup \ldots$

$A_j = \{f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ 0, f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ 1, \ldots, f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ (f_j(n) - 1)\}$ and

$A_{j+1} = \{f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ 0, f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ 1, \ldots, f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ (f_{j+1}(n) - 1)\}$

$\rightarrow A_j\ V_{n-j}(A) = \{f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ 0\ l_1, f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ 1\ l_1, \ldots,$ $f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ (f_j(n) - 1)\ l_1, \ldots\}$ and

$A_{j+1}\ V_{n-(j+1)}(A) = \{f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ 0\ l_1, f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ 1\ l_1, \ldots, f_1(n)$ $f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ (f_{j+1}(n) - 1)\ l_1, ..\}$

Let $label(v_1) = f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ 0\ l_1$, and $label(v_2) = f_1(n)\ f_2(n) \ldots f_{j-1}(n)\ f_j(n)\ 0\ l_1$.

$v_1 \in A_j\ V_{n-j}(A)$ and $v_2 \in A_{j+1}\ V_{n-(j+1)}(A)$ and $H(v_1, v_2) = 1$.

Therefore there is an arc between the vertices v_1 and v_2 and as a consequence the graphs induced by $A_j\ V_{n-j}(A)$ and $A_{j+1}\ V_{n-(j+1)}(A)$ are connected by at least that arc. □

Combining part1 and part2 prooves the connectivity of the graph in the class of HLRN.

Connectivity can also be proven by showing that all HLRNs embed a tree rooted at 0^n as we will see in chapter 5.

4.2 Diameter

The Diameter of HLRN is n meaning that it is always possible to establish a path between any two nodes in an HLRN in a maximum of n steps.

Proposition 3.1: The diameter of an HLRN generated by $A = f_1(n)\ f_2(n) \ldots f_k(n)$ is bounded above by n.

Chapter 4

Routing of Linear Recursive Networks

Once the LRN has been established (node set and edge set), routing and embedding algorithms need to be developed before the network can be functional. Routing algorithms are used to transmit messages between processing elements (nodes), while embedding algorithms are used for partitioning the network into sub-networks so that the different jobs can be conducted in parallel. More over embedding allows applications that run on one type of network to run on a different as long as the former can be embedded in the latter. Also, embedding can be used to reconfigure the network dynamically in case of failures or other factors. This chapter will define several routing algorithms that are defined for all LRN. We will call these algorithms *generic* algorithms since none of these algorithms will be edge set dependent. That is the algorithms are not in function of the interconnection functions. Verification of correctness for each algorithm will be provided along with the complexity analysis. The chapter will end with other algorithms that more specific to a sub-class of the large class of LRN. Embedding algorithms will be presented in chapter 5.

A routing algorithm serves the function of providing the necessary path of communication between one or more processing elements. There are three kinds of routing algorithms: static, semi-dynamic, and dynamic.

Static algorithms require information about the entire network. These algorithms are run off-line. Routes are statically determined and tables are stored and used later to route messages between processing elements. These kinds of algorithms perform well once the network is functional since only tables' lookups are required, but they require storage, which is sometimes not feasible. The network can not be reconfigured dynamically because these tables will no longer contain correct information about the interconnection state of the network. The latter does not constitute a major problem, since network while being reconfigured usually go off-line, and therefore the algorithms can be rerun to determine routes for the new interconnection state.

Semi-Dynamic algorithms are those algorithms that use static information about the network only once to figure out a way to route messages such as computing routing functions. Once the system is functional, these algorithms do not require any substantial storage, nor do they require table lookups. These algorithms no longer need the static information, and behave as though they were dynamic algorithms. If the network is to be reconfigured, these algorithms need to be rerun once to figure out a new way of routing messages. This is done off-line after the network has been reconfigured.

Dynamic algorithms are those algorithms that do not require any global information about the network. These algorithms are usually complicated. They usually utilize some property of the network to figure out a way to route messages. These algorithms are defined for a given topology and therefore are can not be applied for any other topology. This does not apply in general to static and semi-dynamic, as we will see in this chapter.

1 Static algorithms

Static algorithms are very general in that they can be applied to a variety of LRN. They usually require memory to store information to be utilized by the network once it is functional. Network specialist could argue that this is a draw back in using static algorithms. Moreover, they do not adjust to change that occur to the network dynamically such as dynamically reconfiguring the network.

1.1 Dijkstra's algorithm

The first algorithm that we will present is a static routing algorithm, which disposes of all info about the network (node set and edge set). It statically calculates the shortest paths from every node to all other nodes in the network using the well-known optimal shortest path algorithm: Dijkstra's greedy algorithm [6] that runs in O (N^2).

Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted graph G = (V, E) for the case that all weights are non-negative. In our case, G is the graph of the LRN, V is the node set, and E is the edge set. Dijkstra's algorithm does not utilize the semantics of interconnection. It only needs to know the adjacent nodes, for which it keeps an adjacency matrix *adj*. The adjacency matrix is defined as follows:

$$Adj_{ij} = \begin{Bmatrix} 1 & (i,j) \in E_n(A) \\ 0 & Otherwise \end{Bmatrix}.$$

The algorithm also utilizes a weight matrix. In our case it is defined as follows:

$$w_{ij} = \begin{Bmatrix} 0 & if\ i = j \\ 1 & if\ \mathrm{i} \neq j\ and\ (i,j) \in E_n(A) \\ \infty & if\ i \neq j\ and\ (i,j) \notin E_n(A) \end{Bmatrix}$$

A distance value *d[v]* is maintained for each vertex as a shortest path estimate form a source to the vertex v. A path P_1 is estimated to be shorter than path P_2 if and only if the sum of the weights along P_1 is less than the sum of the weights along P_2. Since all weights, in our case, are 1, d [v] also represents the number of hops from source to some vertex v, therefore giving the length of the shortest-path form source to v. And since that length is minimal (shortest), a routing algorithm utilizing Dijkstra's algorithm will route messages optimally.

Along with d[v], a predecessor $\pi[v]$ is maintained for each vertex v. π[v] is the predecessor of v along the shortest path from source to v, and the number of hops to get to v (weight of path) is d[v].

Dijkstra's Algorithm is a greedy algorithm, which works as follows. It first initializes all vertices' distance values to ∞ except source, which gets zero. It also initializes all the predecessor to NIL (No predecessor):

```
INITIALIZE(G, s)
for (int i =0;  i < sizeof(V); i++)
        d[v] = ∞
        π[v] = NIL
d[s] = 0
```

Then it initializes a set S to be used to hold all those vertices that have been already processed. Another set Q is initialized to the node set of the graph. The algorithm repeatedly extracts the vertex with the minimum distance value, from Q, puts the vertex in S, and starts relaxing all its adjacent vertices (not in S). Relaxing an edge simply means that d[v] is updated to contain the smaller value of d[v] and d[u] + w(u, v) where u is an adjacent vertex to v not in S. The predecessor matrix is also subject to change in the relaxation process:

```
RELAX (u, v, w)
        If ( d[v] > d[v] + w (u, v)) then
                d[v] = d[v] + w (u, v)
                π[v] = u
```

The algorithm terminates once Q is empty. Dijkstra's algorithm could written as follows:

```
DIJKSTRA (G, w, s)
        INITIALIZE(G, source)
        S = Empty
        Q = V
while  Q ≠ Empty
```

u = EXTRACT-MIN(Q)

$S = S \cap \{ u \}$

for all adjacent $v \in adj[u]$

RELAX (u, v, w)

Runtime Analysis of Dijkstra's algorithm:

INITIALIZE is a linear process that goes through every element in the vertex set. Therefore $T(n_{INITIALIZE}) = O(n)$. INITIALIZE is not embedded inside other loops, and can not therefore worsen the runtime of Dijkstra's algorithm.

RELAX is obviously a constant procedure and therefore $T(n_{RELAX}) = O(1)$

EXTRACT-MIN is a procedure that extracts the vertex (from Q) with the least distance value. Q contains those vertices not yet processed. Clearly, EXTRACT-MIN can not do worse than finding the smallest of the elements in Q, and that is going though every element in Q, which will take n steps. $T(n_{EXTRACT\text{-}MIN}) = O(n)$

Dijkstra's algorithm calls INITIALIZE, then wraps EXTRACT-MIN and a for loop (for all adjacent $v \in adj[u]$) in a loop of n steps (while $Q \neq$ Empty). We will denote by $T(n_{for\text{-}Loop})$ the time it takes to process the inner for loop (for all adjacent $v \in adj[u]$), which is clearly linear since the number of adjacent vertices is less than n. And we therefore get $T(n_{for\text{-}Loop}) = O(n)$. The total runtime of Dijkstra's can be determined by the following equation:

$$
\begin{aligned}
T(n_{Dijkstra}) &= T(n_{Initialize}) + n(T(n_{Extract-Min}) + T(n_{for-loop})) \\
&= O(n) + n(O(n) + O(n)) \\
&= O(n) + O(n^2) \\
&= O(n^2)
\end{aligned}
$$

To generate all the routing info we need to wrap Dijkstra in loop so that the algorithm can be run for every vertex as being the source, and the minimum distances will be determined, and stored in tables. Hash tables can be used to store the routing info so that future lookup will be constant. The effect of wrapping Dijkstra in a linear loop will lead to a n^3 algorithm. Let T be a hash table. Routing tables can be constructed out of the predecessor vector as follows:

```
for (int i = 0;  i < sizeof (V)  ; i++)

v = V_i

Predecessor = π [n]

While (Predecessor ≠ nil){

        Hash (T, v, predecessor);

        Predecessor = π (predecessor);
```

When the network is functional and a processing element PE_I wants to route a message to PE_j, PE_I will do a hash table lookup and extract the shortest path from PE_i to PE_j.

This routing algorithm presents optimal routes, but requires space to be allocated to keep track of the tables. On the other, this routing algorithm provides an easy to implement procedure to send messages from any source to any destination using the shortest path. The algorithm can also produce alternative paths simply by reordering the vertex set before submission to Dijkstra's algorithm. We will talk about this algorithm some more when we get to the alternative paths' section.

2 Semi-Dynamic algorithms

Semi-Dynamic algorithms are those algorithms that use static information about the network only once to figure out a way to route messages such as computing routing functions. Once the system is functional, these algorithms do not require any substantial storage, nor do they require table lookups. These algorithms no longer need the static information, and behave as though they were dynamic algorithms. If the network is to be reconfigured, these algorithms need to be rerun once to figure out a new way of routing messages. This is done off-line after the network has been reconfigured.

Semi-Dynamic algorithms require static information. A static algorithm such as Dijkstra's algorithm can provide this static information. For our next algorithm, information will be represented like a truth table, where there are inputs and outputs. The inputs will represent the source and destination nodes, while the outputs will denote the first closest stop in the path from the source to the destination.

2.1 An application to switching theory: Switched Algorithm

S	Y	F
0	0	0
0	1	0
1	0	1
1	1	1

In the above table X and Y represent the inputs while F represents the output. The truth table defines the input-output relationship. The table is then translated to mathematical notations (equations) and simplified to obtain a well-defined input-output relationship. Switching theory present the necessary tools to do such a task. A traditional method is to read out the non-zero outputs as follows and further simplify the resulting equation using Boolean algebra rules as follows:

$$\begin{aligned} F &= (\bar{X} \bullet Y) + (X \bullet \bar{Y}) + (X \bullet Y) \\ &= X + Y \end{aligned}$$

More sophisticated translation and simplification methods are documented in switching theory literature such as Karnaugh map and Quine-McCluskey [25, 21].

We will use the above notions to define input-output relationships between source and destination as being inputs, and the first hop along the path from source to destination as being the output. Using a truth table, we will well-define this relationship. We will then obtain a formal mathematical input-output relationship as in above example, using switching theory techniques. We will call these relations routing functions. Each one of these functions defines the next state in the path from source to destination.

A static algorithm will generate the truth tables that we will use. After simplification of the tables, we will obtain the routing functions that can be used to route messages from source to destination. This algorithm is best understood by means of example.

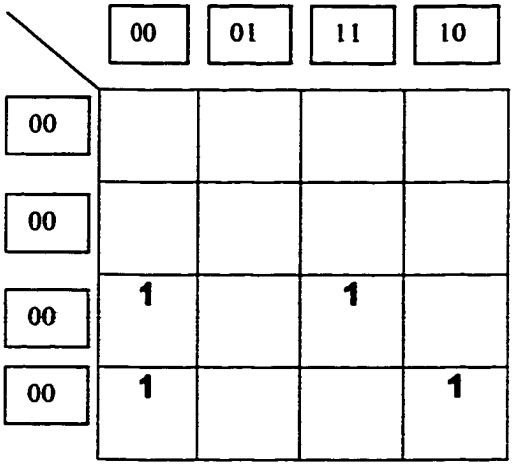
Consider a LRN G = (N, E) where N = {00, 01, 10, 11} and E = {(00, 01), (00, 10), (01, 11), (10, 11)}. Suppose that we input G to Dijkstra's algorithm and we obtain the following truth table:

S_1	S_0	D_1	D_0	X_1	X_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	1

In the above table, S_1S_0 constitute the label of the source nodes, D_1D_0 constitute the labels of the destination nodes and X_1X_0 constitute the first node along the shortest path between source and destination. We will first get the raw mathematical expression equivalent to the above table, and then simplify the expression to its minimal equivalent.

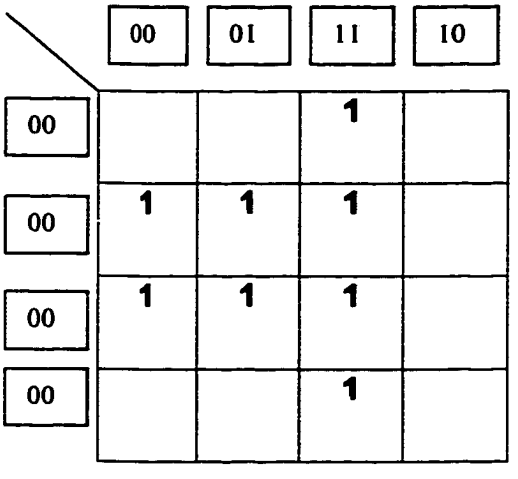
The resulting function X_0 and X_1 will the routing functions used to route messages between source S and destination D. Using K-Map we obtain:

X_1:



$X_1 = S'_1 S'_0 D_1 + S'_0 S_0 D'_0 + S_1 S_0 D_1 D_0$.

X_0:



$X_0 = S'_1 D_0 + S_1 S_0$.

Now $X = X_1X_0$ is always the first hop along the shortest path from $S = S_1S_0$ to $D = D_1D_0$.

Having pre-computed the routing function for an interconnection network statically, the route between any two nodes in the topology can be pre-computed dynamically at the source follows:

Route = S

Intermediate = S

While S ≠ D do

Intermediate $= X_1$ (intermediate, D) + X_0 (intermediate, D)

Route = Route + intermediate

The resulting route is the shortest path from source S to destination D.

Runtime Analysis of switching algorithm

The algorithm utilizes Dijkstra's algorithm to build information about the network, or maybe the network designer can specify that information. It then moves on to generate functions that will be used to route messages by translating the truth table into mathematical expressions and finally simplifying them to get minimal computations while routing messages. Dijkstra's algorithm is an $O(n^3)$ algorithm as was shown in the previous section. Translations and simplification of truth tables is exponential, which them makes the entire process an exponential process. There are commercial packages used in the industry to simplify Boolean expression. These packages are usually used to

assist in building gates, and a variety of electronic devices. These packages are well optimized and allow simplifications to be performed in reasonable time constraints.

The big advantage of this algorithm is that it can be used to route messages in two distinct ways:

Routes can be pre-determined at the source: The route can be attached to the messages, and along the way the message is switched accordingly. The next stop can be determined at every node along the way, by executing the routing function at every node along the way. The advantage in doing it this way is that no additional space is required to store the route in the message. A message carries the address of the destination. The receiving node examines the destination address. If it is different than its address, it re-computes the next hop by setting itself to be the source.

Ease of implementation and speed of execution: Since we are dealing with Boolean functions, it is possible to implement a logical circuit to handle routing since truth tables give rise to logical circuits. This routing strategy no longer needs the static information once the routing functions have been determined, or the logical circuits have been implemented. Its storage requirements are very low.

2.2 Tree algorithm

This routing algorithm is a more specialized routing algorithm that can only be applied to the members of HLRN (Definition 3.8). This algorithm assumes that every node has a local copy of the node set of the graph of the LRN.

The algorithm routes messages from a source S to a destination D by utilizing the services of a coordinator, which will be the root of a tree. Each HLRN can be seen as a tree rooted at the node with the zero address. If two nodes in the network want to exchange messages, the messages can go up to the root of the tree that is the zero labeled node, and then go down the tree towards it destination. The tree structure is well defined as will be proven soon, and the routing strategy is very simple. On the downside, it is not optimal. The path messages traverse is longer than the optimal path that is n steps long. We will shortly prove that the minimum number of hops required to route in a HLRN is n and that this routing strategy requires 2n steps. This routing strategy has several attractive advantages: it does not require much storage, it easy to implement, and it will always take 2n steps to go from point A to point B, so that this strategy can be used to provide a multi-stage implementation of HLRN.

Lemma 4.1: for all $v \in V_n(A)$, there exists a path ,*P*, form v to 0^n.

Proof: (See lemma 3.2) □

Tree Algorithm

To route a message in this scheme from S to D, a route is computed by first finding the route (path) P_1 from S to 0^n, second finding the route (path) P_2 from D to 0^n, and finally the route P is P_1P_2. The algorithm can be expressed as follows:

Stemp = S

Dtemp =D

$P_1 = \{S\}$

$P_2 = \{D\}$

While Stemp $\neq 0^n$

i = position of right most non zero element in Stemp

Stemp = Stemp with a zero at the ith position

$P_1 = P_1 \cup \{Stemp\}$

While Dtemp $\neq 0^n$

i = position of right most non zero element in Dtemp

Dtemp = Dtemp with a zero at the ith position

$P_1 = \{Stemp\} \cup P_1$

$P = P_1 \cup P_2$ // this is the (route) path from S to D.

Correctness of Tree Algorithm

See Lemma 3.2, lemma 3.3, and lemma 3.4 □

Runtime analysis of Tree Algorithm

The algorithm consists of two loops, each of which performs 3 operations. The first is a search through the label for the first non-zero base-r numeral in the label. This search is a linear search that takes at most n steps. Following this search are 2 constant operations. Therefore, each iteration of the loop takes at most n+2 operations. The number of iterations can not exceed n that is the number of base-r numeral in the label. Therefore, each loop can perform at most n (n+2) operations = O (n^2). The two loops are identical so that the above analysis can be applied to both loops. Finally the algorithm perform a constant operation. The total runtime of the algorithm is:

$$\begin{aligned} T(n) &\le n(n+2)+n(n+2)+1 \\ &= 2n^2+4n+1 \\ &\le 3n^2, \forall n \ge 3 \\ \rightarrow T(n) &= O(n^2) \end{aligned}$$

Lemma 4.2: The length of the path P from S to D, denoted by length (P), is bounded above by 2n.

Proof:

First we must mention that we need not show existence of a path, since we have formally proved that the graph of a HLRN is a connected graph in lemma 3.2. We proceed to prove that length(P) $\le$ 2n.

The tree algorithm proceeds by changing the base-r numerals in the label of the node until 0^n is reached. This process is done twice, once to get from the source to 0^n determining the path P_1, and then it is repeated to get from 0^n to the destination determining the path P_2. P the path from source to destination is $P_1 + P_2$ (actually it is P_1

followed by the reverse of P_2). The max number of base-r numerals that need to be changed to traverse P_1 is n, since there are exactly n base-r numerals in the nodes' labels. This means that the maximum number of hops needed to traverse the path P_1 is $\leq$ n. The same argument applies to P_2 and it therefore it takes at most n hops to traverse path P_2. The length of the path P is equal to the length of the length of the path P_1 plus the length of the path P_2. So we get:

$$\begin{aligned} Length(P) &= Length(P_1) + Length(P_2) \\ &\leq n + n \\ &\leq 2n \end{aligned}$$

The longest path between any two nodes using the tree algorithm is 2n.□

A consequence of Lemma 4.2 shows is that a path is a HLRN is also possible in no more than 2n steps → the diameter of HLRN is bounded above by 2n.

As an example, consider $A = 2.1.\ J = \{1,2\} \rightarrow A_1 = \{0, 1\}$ *and* $A_2 = \{20\}$. Let's construct the node set of the LRN generated by $A = 21$.Let's generate the node set $V_3(A)$. Note that $V_n(A) = A_1 V_{n-1}(A) \cup A_2 V_{n-2}(A)$.

$$\begin{aligned} V_0(A) &= \varnothing \\ V_1(A) &= A_1 V_0(A) \cup A_2 V_1(A) \\ &= \{0,1\} \circ \varnothing \cup \varnothing = \{0,1\} \\ V_2(A) &= A_1 V_1(A) \cup A_2 V_0(A) \\ &= \{0,1\} \circ \{0,1\} \cup \{20\} \circ \varnothing \\ &= \{00,01,10,11,20\} \\ V_3(A) &= A_1 V_2(A) \cup A_2 V_1(A) \\ &= \{0,1\} \circ \{00,01,10,11,20\} \} \cup \{20\} \circ \{0,1\} \\ &= \{000,001,010,011,020,100,101,110,111,120,200,201\} \end{aligned}$$

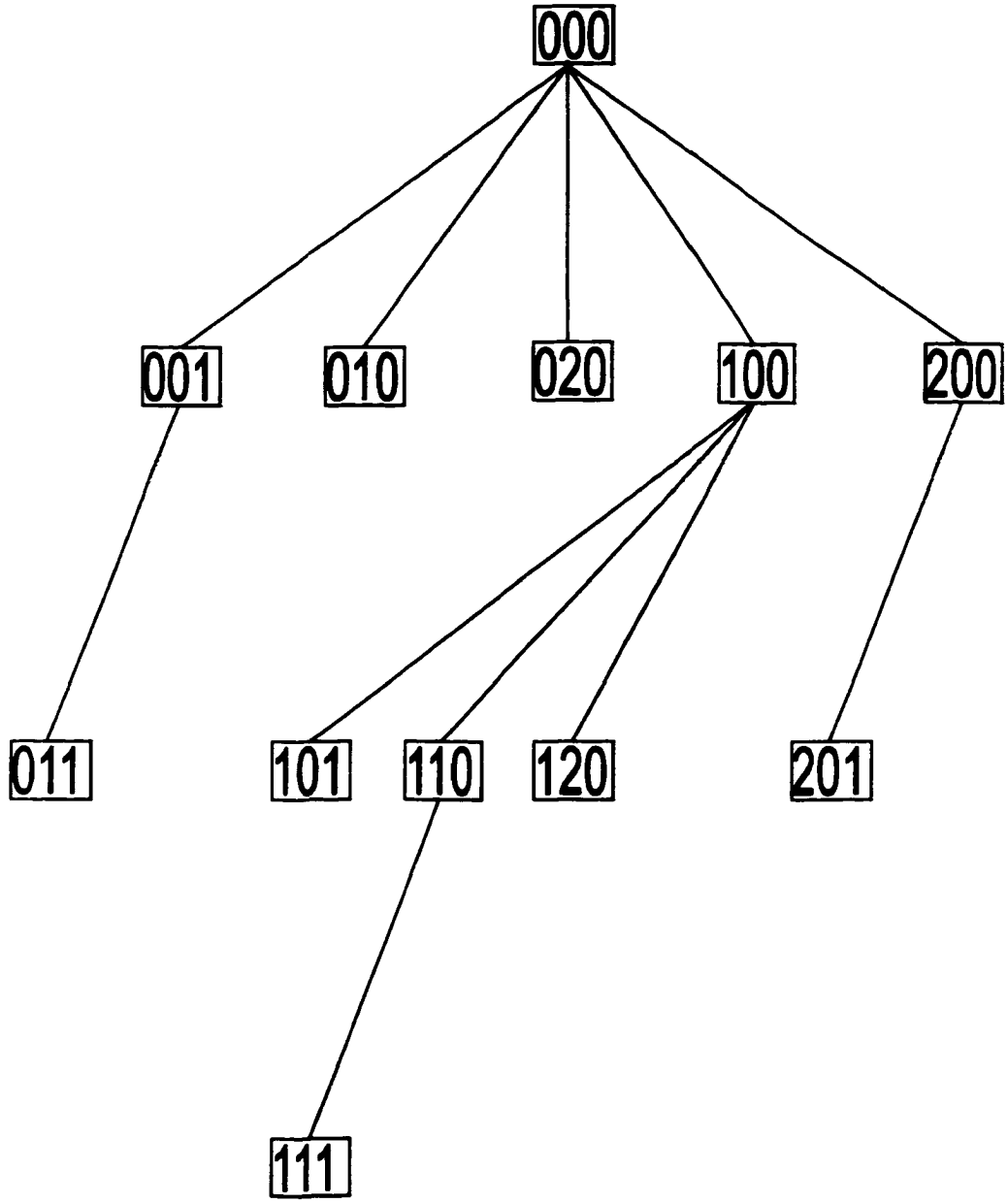


Figure 25: Tree Rooted at 000 that provides routing in LRN generated by A=2.1.

2.3 Transition algorithm

This routing algorithm is a specialized routing algorithm that can only be applied to the members of HLRN (Definition 3.8).

When a node S want to route a message to another node D, the route, r, is pre-determined at S. As the message is travelling form S to D, it will use this information to determine the nest hop along the shortest path. We will first define several operators that will help us demonstrate the algorithm. We assume here that the labels of the nodes are composed of binary elements.

Definition 4.1: Let T be a tertiary operator. T: $V_n(A)$ x N x $V_n(A)$ → X where A is the generator of a LRN, $V_n(A)$ is the node set, and X is the set of all possible binary permutations of the labels of the nodes in $V_n(A)$. T is defined as follows:

$T(S, i, D) = S_{n-1}\ldots S_{I+1}D_IS_{I-1}\ldots S_1S_0$.

The operator T uses S, i, and D to generate a new label. This new label is not necessarily in V_n (A). The new label will only be in $V_n(A)$ if and only if $G_n(A)$ is a complete graph, that is the size of the nodes in the LRN is $N = 2^n$, in which case the vertex set will contain all possible combinations of the components of A forcing every label generated by T to be in $V_n(A)$.

A simple case would be the Hypercube which consists of exactly 2^n nodes, and the node are interconnected if and only if the labels of the nodes differs in one label as is the case here. Routing in a Hypercube is very simple since every combination of 0's and 1's exists. The hamming distance is determined, and messages are routed according to the first one encountered in the hamming distance symbol and so forth. This routing algorithm is optimal and only works because the network is complete. The transition

routing algorithm is a generalization of the algorithm that can be applied to complete and incomplete cubes.

Since the network is incomplete, the routing algorithm is obviously more complicated. As the routing algorithm is searching for the shortest path form S to D, it has to make sure that all the nodes along the way are in $V_n(A)$. The local copy of the vertex set (node set) will be used exactly for that purpose. To find the shortest route (path) from S to D, we will first calculate the hamming distance H(S, D) between S and D, and use T to generate a new label along the shortest path from S to D. We will then verify that that new label is in the vertex set by examining the local copy of the node set. If it is, we the algorithm process into finding the next closed node. Otherwise, We need to verify the existence for another neighbor. The hamming distance is used to generate the possible neighbors, as is the case in the Hypercube. Except that in our case, that neighbor might not exist. The hamming distance is made up of 0s and 1s. The ones indicate that the 2 corresponding labels are different. For example Let S = 011 and D = 110 → H(S, D) = 101, that is that S and D differ in the 0^{th} bit and in the second bit, and the shortest path between them can be no shorter than 2 hops.

Definition 4.2: POS: $\{s|\ s = 0^{i-1}10^{i+1}\}$ → N. POS(s) = i.

Now we are ready to completely describe the algorithm. The route will be stored in a set, r, which is a set of symbols, r_i, containing n-1 0s and a 1, where n is the length of the labels in the LRN.

To calculate the route form source S to destination D, we first the Hamming distance between S and D is determined so that d = H(S, D): d is a sequence of 0s and 1s, where a o indicates that S and D have the same digit at that position, a 1 indicates otherwise. Then, we traverse d form left to right, and when we encounter a 1 at position I, we verify if T(S, I, D), as defined above, belongs to the node set $V_n(A)$. If so, we just found the nearest neighbor to S along the path form S to D, otherwise, we move to the next 1 in D and redo the above-described process. When there are no more zeros in d, the shortest route for S to D has been determined and stored in r. r now contains the necessary input symbols to route the message form S to D by successive applications of r_i to S. Here is the pseudo code of the transition algorithm:

Transition algorithm

Let S and D be source and destination nodes, d will denote the label difference between S and D, r is the set of all routing sequences r_i where r_i are of the form $0^{i-1}10^{i+1}$.

$d = S \oplus D$ $\quad i = 0$ $\quad j = 1$

if S > D swap(S, D)

Repeat

While $d_i = 0$ i++ // search for the first 1

If i = n Terminate // there are no more 1s in d

If $T(S, i, D) \in V_n(A)$

$S = T(S, i, D)$

$r_j = 0^{i-1}10^{i+1}$

j++

$d = d - r_j$

i = 0;

Until (False)

Correctness of the transition algorithm

We have to prove that this routing strategy actually establishes a path between S and D. This requires first proving that there exists a path between S and D for any S and D, and then proving that the way the algorithm decides the next hops is actually correct and will finally get you to the destination. The first part was already proven in lemma 3.1 since this algorithm only applies top HLRN. We only need to show that the way the algorithm is devised, the destination will actually be reached.

The algorithm first computes the label differences between S and D, then uses it to move from S to D. The label difference is a binary string, where a 0 means that S and D have the same base-r numeral at that position, and a 1 means that the base-r numeral differ in that position. The operator T is used to generate possible neighbors. The look up actually confirms the existence of the new node generated, at which instance a hop along the path from S to D has been found. The operation is repeated until there the destination is reached.

Runtime Analysis of the transition algorithm:

The majority of the time spent computing the route is in the repeat statement. There is a linear search for the right most 1, at position k, that is bounded above by n. A comparison is performed to see if the algorithm should terminated, and this is a constant operation that need one unit of time. A new node label is generated in constant time by substituting the kth base-r numeral by d_k. Then, the algorithm executes a table lookup to verify if the newly generated label is actually a valid label. If it is, five constant operations are executed to yield a total runtime of:

$$\begin{aligned} T(n) &\le n(n+1+1+n+5) \\ &\le n(2n+7) \\ &\le n(3n) \\ &\le 3n^2 \\ \rightarrow T(n) &= O(n^2) \end{aligned}$$

Lemma 4.3: The length of the path P from S to D, denoted by length(P) is bounded above by n.

Proof:

To compute the shortest path from S to D, the transition algorithm first computes the hamming distance between S and D. The hamming distance dictates how many base-r numeral are in conflict between the nodes, and also dictates how many hops are needed to get from S to D. Since the hamming distance can not exceed n (there are n base-r

numerals in a node label so that there is at most n different base-r numeral), The length of the path can be at most n.

Lemma 4.4: The transition algorithm computes optimal routes for HLRN.

Proof:

Every new node generated along the way is closer to the destination than it predecessor. Therefore the final route is optimal. □

3 Dynamic algorithms

Dynamic algorithms are those algorithms that do not require any global information about the network. These algorithms are usually complicated. They usually utilize some property of the network to figure out a way to route messages. These algorithms are usually defined for a given topology and can not be applied to other topologies. Designing a generic dynamic algorithm for all the members of the class of LRN is almost impossible.

We will develop some dynamic routing strategies as we define complete topologies in subsequent chapters.

Chapter 5

Containment and Embedding Properties of Linear Recursive Networks

Embedding algorithms developed for a given network topology, or host network, dictate how other topologies are mapped into the host network. It also provides dynamic reconfiguration and the possibility of decomposition of the interconnection network for parallel computation.

Simulation of different topologies is useful when jobs that run efficiently on the host interconnection network need to be run on a different one.

Simulating the interconnection network topology that suits a job best allows for that job to be run efficiently on a variety of other interconnection networks.

In the case of a failure, the system might be reconfigured dynamically. As a result, the system will function on a lower dimension of the network. . Being able to embed several topologies into an interconnection network makes system reconfiguration possible.

An important network design issue is the capability of performing parallel computation on the system. Parallelism allows several jobs to execute simultaneously which results in increasing the performance of the system.

Decomposition of the system makes the parallelism possible. By decomposing the system into several independently functional subsystems, these subsystems can execute different jobs without interfering with one another.

An attractive feature of LRNs is their ease of decomposition. By examining the recurrence equation that models the network, one can easily identifies various subnetworks as to be independent LRNs and part of a larger LRN. For example, the Hypercube generated by $A = 2$ satisfies the linear recurrence equation $X_n = 2n$. Therefore, a cube of dimension C_3 is made of two cubes of dimension C_2. Each C_2 is a LRN in its own right and possess all the topological properties of C_3. Two different tasks can be run on both C_2 in parallel allowing a much higher performance.

Another example is the n-star network that satisfies the linear recurrence relation $X_n = n X_{n-1}$. Every n-dimensional star network, S_n, is made up of n copies of (n-1)-dimensional star networks,S_{n-1}.

1 m-equivalence and Partial ordering of generating sequences

Definition 5.1: A sequence $A = f_1(n).f_2(n)...f_k(n)$ is said to be strictly less than a sequence $B = f'_1(n).f'_2(n)...f'_k(n)$, denoted by A<B, if and only if: $|A| \leq |B|$ and

$f_i(n) = f'_i(n)$ *for all* $1 \leq i < j < k$ and $f_j(n) < f'_j(n)$ and $f_i(n) = 0$ *for all* $j < i \leq k$

Definition 5.2: A sequence $A = f_1(n).f_2(n)...f_k(n)$ is said to be less or equal than a sequence $B = f'_1(n).f'_2(n)...f'_k(n)$, denoted by A≤B, if and only if: $|A| \leq |B|$ and

$f_i(n) = f'_i(n)$ *for all* $1 \leq i < j < k$ and $f_j(n) \leq f'_j(n)$ and $f_i(n) = 0$ *for all* $j < i \leq k$

Corollary 5.1: $(A < B) \rightarrow (A \leq B)$.

Proof:

Follows directly from definitions 5.1 and 5.2. □

Definition 5.3: Let $B = a_1a_2...a_n$ be a sequence of n symbols. A is a prefix of B if:

A is empty.

$A = a_1...a_k$ such that $k \leq n$.

Example:

Let B = 1324 The following is a legal list of prefixes:

A = 13, A = 132, A = 1, A = empty, A = 1324

The following not prefixes of s:

A = 12, A = 134, A = 324, A = 24.

Corollary 5.2: (A is a prefix of B) $\rightarrow A \leq B$

Proof:

Follows directly from definition 5.3

If A is a prefix of $B = b_1 \ldots b_k$, then $A = b_1 \ldots b_m$ where $1 \leq m \leq k \rightarrow A \leq B$. □

Example: Let A = 2.3.2, B = 2.3.1, and C = 2.3. D = 2.0.0.1. Then,

$A \leq A$ is true

$B < A$ is true

$C < A$ is true

$C < B$ is true

$D \leq A$ is false and $A \leq D$ is false. A and D are not related.

Neither is B with D, nor is C with D.

Definition 5.4: An expansion of a sequence $A = f_1(n).f_2(n)...f_k(n)$ is a sequence B such that $|A| \leq |B|$ and $X_n(A) = X_n(B)$ for all $0 < |B| \leq n$ and

$$B = f_1(n) \ldots f_i(n)\text{-}x \ldots f_k(n).0^i + 0^i.x^*(f_1(n).f_2(n)...f_k(n))$$

Example:

Let A = 2.5.8. Then, all the following are legal expansions of A: 2.5.8, 1.7.13.8, and 0.9.18.8.

The following are illegal expansions of A: 1.6.13.8, 1.7.13.7, and 0.8.18.8

Lemma 5.1: (B is an expansion of A and B ≠ A) → |B| > |A|

Proof:

B is an expansion of A → $B = f_1(n)\ldots f_i(n)\text{-}x\ldots f_k(n).0^i + 0^i.x^*(f_1(n).f_2(n)\ldots f_k(n))$

x = 0 → $B = f_1(n)\ldots f_i(n)\ldots f_k(n).0^i = f_1(n)\ldots f_i(n)\ldots f_k(n) = A$

Trailing zeros in a generator are unimportant. But A ≠ B → x can not be zero.

Since x ≠ 0 → $B = f_1(n)\ldots f_i(n)\text{-}x\ldots f_k(n).0^i + 0^i.x^*(f_1(n).f_2(n)\ldots f_k(n))$

→ |B| = k + I

We know that |A| = k < k + i, since $1 \leq i \leq k$

→ |A| < |B|. □

Definition 5.5: A sequence B is said to be a mutation of the generator $A = f_1(n).f_2(n)\ldots f_k(n)$ if and only if:

- B is an expansion of A.
- B is obtained from an expansion C of A by subtracting a base-r numeral from a component of C.
- All expansions of mutations.
- Nothing else.

Example:

Let A = 2.5.8 all the following are legal mutations of A:

2.5.8, 1.7.13.8, 0.9.18.8, 2.4.8, 1.5.8, 1.5.7, 1.7.13.8, 1.7.2.7, 1.6.12.8...

The following are illegal mutations of A: 2.5.9, 2.6.8, 2.6.7, 1.8.13.8...

Definition 5.6: Let $\subseteq$ be a relation defined on the generators of LRNs as follows:

Let A, B be generating sequences for a LRN.

Let $A = f_1(n).f_2(n)...f_k(n)$ and $B = f'_1(n).f'_2(n)...f'_{k'}(n)$

We say A contains B, or B is derived from A, and denote it by $A \supseteq B$ or $B \subseteq A$ if and only if B is a mutation of A.

Example: Let *A = 2.3.4, B = 1.5.7.4, C = 1.5.7.3, D = 1.4.8.8.7.3*

X_n is genrated by A:

$$X_n = 2X_{n-1} + 3X_{n-2} + 4X_{n-3}$$
$$= X_{n-1} + X_{n-1} + 3X_{n-2} + 4X_{n-3}$$

We substitue for $X_{n-1} = 2X_{n-2} + 3X_{n-3} + 4X_{n-4}$

We get:

$$X_n = X_{n-1} + 2X_{n-2} + 3X_{n-3} + 4X_{n-4} + 3X_{n-2} + 4X_{n-3}$$
$$= X_{n-1} + 5X_{n-2} + 7X_{n-3} + 4X_{n-4}$$

Therefore B = 1.5.7.4 is a mutation of A.

and thus $A \supseteq B$

$A \supseteq$ C since C = B - 0.0.0.1 and B is a mutation of A.

$A \supseteq D$ since D is an expansion of C

We can Also obtain D from an expansion of B : D = 1.4.8.9.7.4 - 0.0.0.1.0.1

In the above example, A was first characterized, and B was obtained from A by simple algebraic manipulations. It should be rather clear that in the above example, forming B from A is not possible. There is no way to make backward substitutions, if A is not known.

Note that $\subseteq$ is not a symmetric relation.

Lemma 5.2: $\subseteq$ is a partial ordering.

Proof:

Let $A = f_1(n).f_2(n)...f_k(n)$, $B = f'_1(n).f'_2(n)...f'_{k_1}(n)$, $C = f''_1(n).f''_2(n)...f''_{k_2}(n)$

To prove that the relation $\subseteq$ is a partial ordering, we have to prove that $\subseteq$ is reflexive, anti-symmetric and transitive.

Reflexive

Since A is an expansion of A → A is a mutation of A → A $\subseteq$ A.

Thus, $\subseteq$ is a reflexive relation.

Anti-symmetric

Let A, B be generating sequences. Furthermore, assume that B $\subseteq$ A and A $\subseteq$ B.

B $\subseteq$ A → B is a mutation of A → either B = A, or B is an expansion of A but not A, or B = C – s, for some expansion, C, of A, some s $\leq$ C and B $\neq$ A.

$A \subseteq B \rightarrow$ A is a mutation of B $\rightarrow$ either $A = B$, or A is an expansion of B but not B, or $A = D - r$, for some expansion, D, of B, some $r \leq D$ and $A \neq B$.

Case1: B is an expansion of A but not equal to A.

By Lemma 5.1, $|B| > |A| \rightarrow$ by definition 5.4 and definition 5.5, A can not be a mutation of $B \rightarrow A \subseteq B$ is false. But we assumed that $A \subseteq B \rightarrow$ (B is an expansion of A but not A) can not happen if $B \subseteq A$ and $A \subseteq B$.

Case2: A is an expansion of B but not equal to B.

Same argument as case 1. A is an expansion of B but not B can not happen if $B \subseteq A$ and $A \subseteq B$.

Case3: $B = C - s$, for some expansion, C, of A, some $s < C$ and $B \neq A$.

Since $B \neq A$ either $C = A$ and $s \neq 0$, or C is an expansion of A but not A

$B \subseteq A$ and $C = A \rightarrow |B| = |C| = |A|$. Also $B < C \rightarrow B < A \rightarrow$

$A \subseteq B \rightarrow$ A is a mutation of B and since $|A| = |B|$, we must have $A < B$. Contradiction!

Therefore, $B = C - s$, for some expansion C of A, some $s < C$ and $B \neq A$ and $C = A$ can not happen.

If $A \neq C$, it is a similar case to case1.

Case4: $A = D - r$, for some expansion D of B, some $r < D$ and $A \neq B$.

Same as case3.

$\rightarrow$ ($B \subseteq A$ and $A \subseteq B \rightarrow A = B$)

A = B of course →(B $\subseteq$ A and A $\subseteq$ B → A = B) since every sequence is a mutation of itself.

→ (B $\subseteq$ A and A $\subseteq$ B ←→ A = B)

→ $\subseteq$ is anti-symmetric.

Transitive

Let A, B and C be 3 distinct generating sequences such that A $\subseteq$ B and B $\subseteq$ C.

We need to prove that C $\subseteq$ B and B $\subseteq$ A →C $\subseteq$ A. That is $\subseteq$ is transitive.

$$C \subseteq B \rightarrow X_n = f'_1(n)X_{n-1} + \ldots + f'_{k_1}(n)X_{n-k_1} = f''_1(n)X_{n-1} + \ldots + f''_{k_2}(n)X_{n-k_2}$$

$$B \subseteq A \rightarrow X_n = f_1(n)X_{n-1} + \ldots + f_k(n)X_{n-k} = f'_1(n)X_{n-1} + \ldots + f'_{k_1}(n)X_{n-k_1}$$

$$\begin{aligned} X_n &= f_1(n)X_{n-1} + f_2(n)X_{n-2} + \ldots + f_k(n)X_{n-k} \\ &= f'_1(n)X_{n-1} + f'_2(n)X_{n-2} + \ldots + f'_{k_1}(n)X_{n-k_1} \\ &= f''_1(n)X_{n-1} + f''_2(n)X_{n-2} + \ldots + f''_{k_2}(n)X_{n-k_2} \end{aligned}$$

$$X_n = f_1(n)X_{n-1} + \ldots + f_k(n)X_{n-k} = f''_1(n)X_{n-1} + \ldots + f''_{k_2}(n)X_{n-k_2}$$

$C \subseteq A$. □

For a given generator, we can derive infinitely many sequences that satisfy the partial ordering relation.

For example, for A = 2, (B = 04) $\subseteq$ A, (C = 008) $\subseteq$ A. $X_n = 2\ X_{n-1}$ (A=2) can be rewritten in a variety of ways such as:

$X_n = 4\ X_{n-2}$ → B = 04 or $X_n = 8\ X_{n-3}$ → C = 008.

The recurrence equation has the same solution for n = 3.

$X_n = 2\ X_{n-1} \rightarrow X_3 = 2 * 2^{3-1} = 2 * 4 = 8.$

$X_n = 4\ X_{n-2} \rightarrow X_3 = 4 * 2^{3-2} = 4 * 2 = 8.$

$X_n = 8\ X_{n-3} \rightarrow X_3 = 8 * 2^{3-3} = 8 * 1 = 8.$

In the case that the recurrence equation has the same solution for some m we say that the generators are m-equivalent.

Definition 5.7: Let A and B be generators of an LRN. We say A and B are m-equivalent and denote it by $A \equiv_m B$ if and only if B $\subseteq$ A and $X_n(A) = X_n(B)$ *for all* $n \geq m$.

Lemma 5.3: Let $A = f_1(n).f_2(n)...f_k(n)$ and let

$B = f'_1(n).f'_2(n)...f'_{k'}(n) = 0.f_2(n).f_3(n)...f_{k'}(n).0 + 0.(f_1(n) * A)$

$\rightarrow A \equiv_{k+1} B$

Proof:

$B \subseteq A$

$B = 0.f_2(n).f_3(n)...f_{k'}(n).0 + 0.(f_1(n)*A)$

Let $x = f_1(n) \rightarrow$

$B = 0.f_2(n).f_3(n)...f_{k'}(n).0 + 0.(x*A)$

$= (x\text{-}x).f_2(n).f_3(n)...f_{k'}(n).0 + 0.(x*A)$

$= (f_1(n)\text{-}x).f_2(n).f_3(n)...f_{k'}(n).0 + 0.(x*A)$

$= (f_1(n)\text{-}x).f_2(n).f_3(n)...f_{k'}(n).0^i + 0^i.(x*A)$ *for* $i = 1$

$= f_1(n)... f_i(n)\text{-}x... f_k(n).0^i + 0^i.x*(f_1(n).f_2(n)...f_k(n))$

$\rightarrow$ *B is an* expansion *of A*

$\rightarrow$ *B is a mutation of A*

$\rightarrow B \subseteq A$

$A \equiv_{k+1} B$

We proved in part (a) that B was a mutation of A because B was an expansion of A. By Definition 5.4, $X_n(A) = X_n(B)$ for $n \geq |B| = k+1$. □

A consequence of lemma 5.3 is that we can derive indefinitely many sequences that are m-equivalent to a sequence A and each of these sequences carries additional information about the original network. The generator A models an interconnection network whose graph is $G_n(A)$. The base-r numeral components of A tell us of how many lower dimension graph is $G_n(A)$ made of. So if $A = 2.3$, $G_n(A)$ is made of two copies of $G_{n-1}(A)$ and three copies of $G_{n-2}(A)$. If $A \equiv_{k+1} B$, then $B = 0.7.6$. This implies that the original interconnection network modeled by A is made of seven copies of $G_{n-2}(A)$, information which is not directly provided by A. If we find m-equivalent sequences to B, we will obtain more information about the interconnection network. This information will be used to actually partition and decompose the network.

Next we present an algorithm that will generates m-equivalent sequences for a given sequence by generalizing lemma 5.3. This algorithm will be used to obtain info about all lower dimension networks. The m-equivalent sequences generated by the algorithm will have many leading zeros. The first non-zero entry, scanning the sequence form the left, will reflect the number of lower dimension graphs that contribute to constituting the original graph.

The procedure to find all these sequences is a recursive procedure that takes as input p and can be formulated as follows. We only present the constant coefficient case. The general case follows automatically:

$a_i(p) = a_{(i-p+1)} * a_{(p-1)}(n-1) + a_i(p-1)$, *where* $a_i(1)=a_i$, *and* $a_i(p) = 0 \ \forall i<p$.

p is the dimension of the graph which contributes $a_i(p)$ to the original graph. For illustration, we will store the results in a table. The table will consist of a column, labeled p, representing the position of the non-zero term we are interested in, the rest of the columns will reflect the remaining base-r numeral that constitute the sequences. p = 1, the first row, corresponds to the generator of the interconnection network being analyzed. Subsequent rows are filled according the following recurrence equation described above.

Example: Let $A = 213$, and $1 \leq p \leq 5$

<u>p = 2</u>

$a_1(2) = 0$.

$a_2(2) = a_1 * a_1(1) + a_2(1) = 2*2 + 1 = 5$.

$a_3(2) = a_2 * a_1(1) + a_3(1) = 1*2 + 3 = 5$.

P	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	**2**	**1**	**3**	**0**	**0**	**0**	**0**
2	**0**	**5**	**5**	**6**	**0**	**0**	**0**
3	**0**	**0**	**15**	**11**	**15**	**0**	**0**
4	**0**	**0**	**0**	**41**	**30**	**45**	**0**
5	**0**	**0**	**0**	**0**	**112**	**86**	**123**

$a_4(2) = a_3 * a_1(1) + a_4(1) = 3*2 + 0 = 6$.

$a_5(2) = a_4 * a_1(1) + a_5(1) = 0$.

$a_6(2) = a_5 * a_1(1) + a_6(1) = 0$.

$a_7(2) = a_6 * a_1(1) + a_7(1) = 0$.

<u>p = 3</u>

$a_1(3) = 0$.

$a_2(3) = 0$.

$a_3(3) = a_1 * a_2(2) + a_3(2) = 2*5 + 5 = 15$.

$a_4(3) = a_2 * a_2(2) + a_4(2) = 1*5 + 6 = 11$.

$a_5(3) = a_3 * a_2(2) + a_5(2) = 3*5 + 0 = 15$.

$a_6(3) = a_4 * a_2(2) + a_6(2) = 0$.

$a_7(3) = a_5 * a_2(2) + a_7(2) = 0$.

p = 4

$a_1(4) = 0$.

$a_2(4) = 0$.

$a_3(4) = 0$.

$a_4(4) = a_1 * a_3(3) + a_4(3) = 2*15 + 11 = 41$.

$a_5(4) = a_2 * a_3(3) + a_5(3) = 1*15 + 15 = 30$.

$a_6(4) = a_3 * a_3(3) + a_6(3) = 3*15 + 0 = 45$.

$a_7(4) = a_4 * a_3(3) + a_7(3) = 0$.

p = 5

$a_1(5) = 0$.

$a_2(5) = 0$.

$a_3(5) = 0$.

$a_4(5) = 0$.

$a_5(5) = a_1 * a_4(4) + a_5(4) = 2*41 + 30 = 112$.

$a_6(5) = a_2 * a_4(4) + a_6(4) = 1*41 + 45 = 86$.

$a_7(5) = a_3 * a_4(4) + a_7(4) = 3*41 + 0 = 123$.

We will now device an algorithm that takes as input a generator A, and fills a table, T, with m-equivalent sequences. Each one of these sequences presents unique information about the network being analyzed.

```
Analyze (A)
T[1] = A
 for i = 2 to p do
  for j = i to k do
   T[i][j] = T[1][j-i+1] * T[i-1][i-1] + T[i-1][j]
```

The table, T, contains information about the network being analyzed. Every T[i] represents row which elements constitute an m-equivalent sequence to A. The first non-zero entry in every T[i], T[i][j] corresponds to the number of copies of the graph of dimension i, G_i, that participates in $G_n(A)$.

Lemma 5.4: $\leq$ is a partial ordering on generator sequences

Proof:

Let $A = f_1(n).f_2(n)...f_k(n)$, $B = f'_1(n).f'_2(n)...f'_{k_1}(n)$, $C = f''_1(n).f''_2(n)...f''_{k_2}(n)$

To prove that the relation $\leq$ is a partial ordering, we have to prove that $\leq$ is reflexive, anti-symmetric and transitive.

Reflexive

$A \leq A$ vacuously $\rightarrow \leq$ is Reflexive.

Anti-symmetric

Let A, B be generating sequences. Furthermore, assume that $B \leq A$ and $A \leq B$.

$B \leq A \rightarrow$ either $B = A$ or $B < A$.

The assumption above is that $A \leq B \rightarrow B < A$ can not be the case

$\rightarrow B = A$

Therefore $\leq$ is anti-symmetric

Transitive

Let A, B and C be 3 distinct generating sequences such that $A \leq B$ and $B \leq C$.

We need to show that a consequence of the above is that $A \leq C$. That is $\leq$ is transitive.

$A \leq B \rightarrow |A| \leq |B|$ and $a_i = b_i$ for all $1 < i < j$ and $a_j \leq b_j$ and $a_m = 0$ for all $j < m \leq k$

$B \leq C \rightarrow |B| \leq |C|$ and $b_i = c_i$ for all $1 < i < j$ and $b_j \leq c_j$ and $b_m = 0$ for all $j < m \leq k$

$\rightarrow |A| \leq |C|$ and $a_i = b_i = c_i$ for all $1 < i < j$ and $a_j \leq b_j \leq c_j$ and $a_m = b_m = 0$ for all $j < m \leq k$

$\rightarrow |A| \leq |C|$ and $a_i = c_i$ for all $1 < i < j$ and $a_j \leq c_j$ and $a_m = 0$ for all $j < m \leq k$

$\rightarrow \leq$ is transitive.

$\therefore \leq$ is a partial ordering $\square$

It might intuitively seem that mutations and m-equivalence would directly lead to embedding LRNs. Unfortunately it turns out that the intuition was wrong. Stricter constraints need to be imposed on sequences in order to guaranty embedding. The second partial ordering, $\leq$, will be utilized in the following section to define embedding and containment properties of LRNs. Mutation and m-equivalence can still be utilized to obtain additional information about the network as shown in an example above but are not well suited to study the containment properties of LRNs. The following example illustrates how mutations fail into well defining embedding in LRNs.

Assume that A = 2.1.7.4 and B = 2.1.3.3. According to definition 5.6, B $\subseteq$ A. The seed sets of A and B are:

A_1 = {0, 1}, A_2 = {2.0}, A_3 = {2.1.0, 2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.1.5, 2.1.6}, A_4 = {2.1.7.0, 2.1.7.1, 2.1.7.2, 2.1.7.3, 2.1.7.4} and

B_1 = {0, 1}, B_2 = {2.0}, B_3 = {2.1.0, 2.1.1, 2.1.2}, B_4 = {2.1.3.0, 2.1.3.1, 2.1.3.2}.

The node label v = 2.1.3.2 $\in$ V_4(B) and is obtained from seed set B_4. But v $\notin$ V_4(A) as it is not possible to construct the label of the node v from the seed sets of A.

There are cases where graphs generated by mutations of generators can be embedded in graphs generated by original generators, but we can not generalize them as the above example creates the contradiction.

2 Embedding in LRNs

We have shown in the previous section that there is a partial ordering, $\subseteq$, of LRN generators and we showed that some sequences are equivalent. The property was referred

to as m-equivalence, $\equiv_m$. We also showed that $\leq$ is another partial ordering defined on generator of LRNs.

We will present, in this section, sufficient conditions for LRNs generated by different sequences to embed one another. We will present several algorithms that will allow us to compare generators and relate the underlying graphs in such a way that one can be embedded into another. It should be noted that the embedded graphs must satisfy the same interconnection relation as the original graph. In general, if we define two different interconnection functions for a generator A, the corresponding graphs will not be related. Same thing applies for A and its mutations of its m-equivalent sequences.

Lemma 5.5: Let $A = f_1(n).f_2(n)...f_k(n)$, Then $G_n(A)$ can embed $G_r(A)$ for all $1 \leq r \leq k$.

Proof:

$G_n(A)$ is made of $f_1(n)$ copies of $G_{n-1}(A)$, and $f_2(n)$ copies of $G_{n-2}(A)$,...

$V_{n-r}(A) = \{l_1, ..., l_m\}$for $r < n$. $|l_1| = |l_2| = ...|l_m| = n-r$. So $l_1, ..., l_m \notin V_n(A)$ since if $v \in V_n(A)$, $|v| = n$.

$V_n(A) = \cup A_j \circ V_{n-r}(A)$

I define the function R: $V_{n-j}(A) \rightarrow V_n(A)$ such that $R(v) = a_1...a_{j-1}.0.v$ where $v \in V_{n-j}(A)$.

$v \in V_r(A) \rightarrow |R(v)| = |a_1...a_{r-1}.0.v| = n$ and $(R(v) = a_1...a_{r-1}.0.v) \in A_r \circ V_{n-r}(A)$ since $a_1...a_{r-1}.0 \in A_r \rightarrow R(v) \in V_n(A)$.

So for every $v \in V_{n-r}(A)$, $R(v) \in V_n(A)$.

We have just maped all the elements in $V_{n-r}(A)$ to $V_n(A)$ but simply attaching a sequence as a prefix.

Let $V'_{n-r}(A) = R(V_{n-r}(A)) \rightarrow V'_n(A) \subseteq V_n(A)$.

We need to show now that if v_1 and $v_2 \in V_{n-r}(A)$ and there is an arc from v_1 to v_2, there will be an arc from $R(v_1)$ to $R(v_2)$.

The latter depends on the interconnection function used to draw an arc between the two nodes.

Without loss of generality, we assume here that the nodes are interconnected by means of hamming distance. v_1 is connected to $v_2 \rightarrow H(v_1, v_2) = 1 \rightarrow H(R(v_1), R(v_2)) = 1$ since R put the same prefix for all the vertices it maps from $V_{n-r}(A)$ to $V_n(A) \rightarrow R(v_1)$ is connected to $R(v_2)$.

Thus, connectivity is conserved.

This implies that $G_n(A)$ can embed $G_r(A)$ for all $1 \leq r \leq k$

Lemma 5.6: Let $A = f_1(n).f_2(n)...f_k(n)$ and $B = f'_1(n).f'_2(n)...f'_k(n)$ such that $B \leq A$ $\rightarrow V_n(B) \subseteq V_n(A)$.

Proof:

$B \leq A \rightarrow |A| \leq |B|$ and

$f_i(n) = f'_i(n)$ *for all* $1 \leq i < j < k$ and $f_j(n) \leq f'_j(n)$ and $f_i(n) = 0$ *for all* $j < i \leq k$

We will proceed by strong induction. We will assume that the lemma is true for all node sets with node labels < n

$v \in V_n(B) \rightarrow v \in B_j \circ V_{n-j}(B)$

Base case: $|A| = 1$

$n = 1 \rightarrow V_n(B) = B_1 V_0(B) = B_1$

$n = 1 \rightarrow V_n(A) = A_1 V_0(A) = A_1$

If $a_1 = b_1 \rightarrow A_1 = B_1 \rightarrow V_n(A) = V_n(B) \rightarrow V_n(B) \subseteq V_n(A)$

If $a_1 \neq b_1 \rightarrow b_1 < a_1$ since $B < A \rightarrow B_1 \subseteq A_1 \rightarrow V_n(B) \subseteq V_n(A)$

$B_1 = \{0, 1, \ldots, b_1\text{-}1\}$ and $A_1 = \{0, 1, \ldots a_1\text{-}1\}$

Since $b_1 < a_1$, $A_1 = \{0, 1, \ldots b_1\text{-}1, b_1, \ldots a_1\text{-}1\}$

$\rightarrow B_1 \subseteq A_1$.

Inductive step: $|A| = n$

Assume that $V_j(B) \subseteq V_j(A)$ for all $0 < j < n$

Note that also $V_{n-j}(B) \subseteq V_{n-j}(A)$ since $0 < j < n \rightarrow 0 < n\text{-}j < n$

$v \in B_j \circ V_{n-j}(B)$ for $j > 0$ and $B \leq A$.

if $a_j = b_j \rightarrow A_j = B_j \rightarrow v \in A_j \circ V_{n-j}(B) \rightarrow v \in A_j \circ V_{n-j}(A) \rightarrow v \in V_n(A)$

if $a_j \neq b_j \rightarrow b_j < a_j \rightarrow A_i = B_i \; \forall \; j$: $1 \leq i \leq j\text{-}1$ and $b_j \leq a_j \rightarrow B_j \subseteq A_j$.

Thus, $v \in B_j \circ V_{n-j}(B) \rightarrow v \in A_j \circ V_{n-j}(B) \rightarrow v \in A_j \circ V_{n-j}(A)$ by induction hypothesis.

$\therefore (B \leq A) \rightarrow V_n(B) \subseteq V_n(A)$. □

Lemma 5.7: Let A, and B be two generators such that $B < A$. Then, $G_n(B)$ is a sub-graph of $G_n(A)$, that is $G_n(B)$ can be embedded in $G_n(A)$. We will denote this relation $G_n(A) \subseteq G_n(B)$.

Proof:

$G_n(B) = (V_n(B), E_n(B))$

$B < A \rightarrow$ By lemma 5.6, $V_n(B) \subseteq V_n(A) \rightarrow$ all the nodes in $G_n(B)$ are in $G_n(A)$.

The interconnection method used to generate $G_n(B)$ is the same as the one used to generate to $G_n(A)$.

So, $v_1, v_2 \in V_n(B) \rightarrow v_1, v_2 \in V_n(A)$

Thus, $(v_1, v_2) \in E_n(B) \rightarrow (v_1, v_2) \in E_n(A)$

$\rightarrow E_n(B) \subseteq E_n(A)$

$\rightarrow G_n(A) \subseteq G_n(B)$.

Example: Let A = 2.3, B = 2.1 and C = 2 for n = 2.

(B = 1.3 does not apply to this example, A < B is false)

A = 2.3

$A_1 = \{0, 1\}$, $A_2 = \{2.0, 2.1, 2.2\}$

$V_1(A) = \{0, 1\}$

$E_1(A) = \{(0, 1)\}$

$V_2(A) = \{0.0, 0.1, 1.0, 1.1, 2.0, 2.1, 2.2\}$

$E_2(A) = \{(0.0, 0.1), (0.0, 1.0), (0.0, 2.0), (0.1, 1.1), (0.1, 2.1), (1.0, 1.1), (1.0, 2.0), (1.1, 2.1), (2.0, 2.1), (2.0, 2.2), (2.1, 2.2)\}$

B = 2.1

$B_1 = \{0, 1\}$, $B_2 = \{2.0\}$

$V_1(B) = \{0, 1\}$

$E_1(B) = \{(0, 1)\}$

$V_2(B) = \{0.0, 0.1, 1.0, 1.1, 2.0\}$

$E_2(B) = \{(0.0, 0.1), (0.0, 1.0), (0.0, 2.0), (0.1, 1.1), (1.0, 1.1), (1.0, 2.0)\}$

C = 2

$C_1 = \{0, 1\}$

$V_1(C) = \{0, 1\}$

$E_1 = \{(0, 1)\}$

$V_2(C) = \{0.0, 0.1, 1.0, 1.1\}$

$E_2(C) = \{(0.0, 1.0), (0.0, 1.0), (1.0, 1.1), (0.1, 1.1)\}$

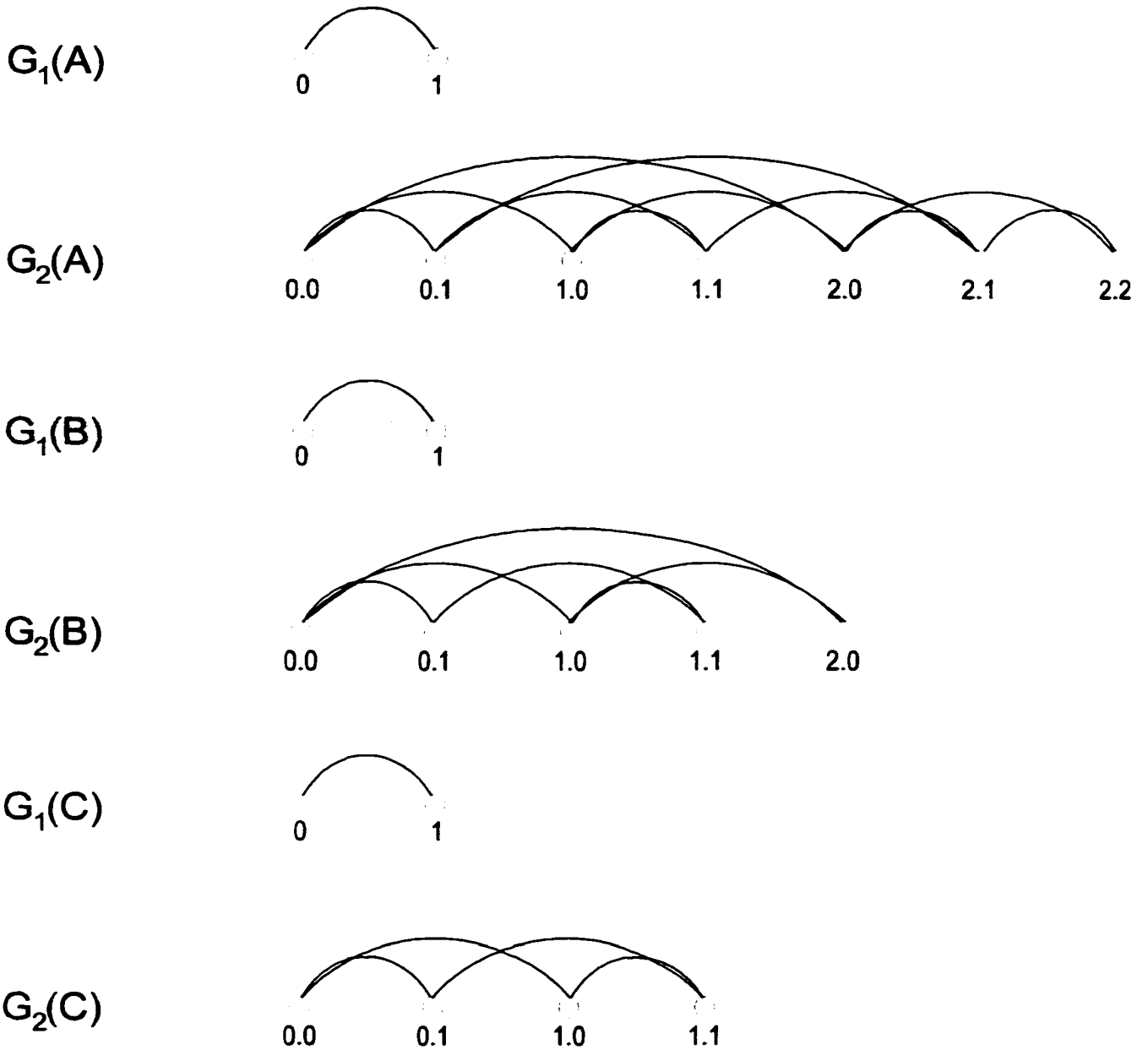


Figure 26: $G_n(C = 2) \subseteq G_n(B = 2.1) \subseteq G_n(A = 2.3)$.

Figure 26 clearly shows that $G_2(B)$ is a sub-graph of $G_2(A) \rightarrow G_2(B) \subseteq G_2(A) \rightarrow G_2(B)$ can be embedded in $G_2(A)$, also $G_2(C)$ is a sub-graph of $G_2(A) \rightarrow G_2(C) \subseteq G_2(A)$ and

$G_2(C)$ can be embedded in $G_2(A)$. No relation ship is yet established between $G_2(B)$ and $G_2(C)$ even if the graph hints that $G_2(C)$ can be embedded in $G_2(B)$.

Lemma 5.8: Let $A = f_1(n).f_2(n)...f_k(n)$ and $B = f'_1(n).f'_2(n)...f'_{k'}(n)$ such that B is a prefix of A. Then $B_j = A_j$ for all $j \leq k'$.

Proof:

B is a prefix of A → $f'_1(n) = f_1(n), f'_2(n) = f_2(n)...f'_{k'}(n) = f_{k'}(n)$

→ $B = f_1(n).f_2(n)...f_{k'}(n)$

$$B_j = \{f'_1(n).f'_2(n)...f'_{j-1}(n).0, f'_1(n).f'_2(n)...f'_{j-1}(n).1,.., f'_1(n).f_2(n)...f'_{j-1}(n)(f'_j(n)-1) : j \in J\}$$
$$= \{f_1(n).f_2(n)...f_{j-1}(n).0, f_1(n).f_2(n)...f_{j-1}(n).1,.., f_1(n).f_2(n)...f_{j-1}(n)(f_j(n)-1) : j \in J\}$$
$$= A_j \text{ for all } j \leq k'$$

→ $B_j = A_j$ for all $j \leq k'$. □

Lemma 5.9: Let A and B be generators for two distinct LRNs such that B is a prefix of A. Then $V_n(B) \subseteq V_n(A)$.

Proof:

The proof is by induction on n.

Base Case: n =1

$V_1(B) = B_1 = A_1 = V_1(A) \rightarrow V_1(B) \subseteq V_1(A)$ and thus $V_n(B) \subseteq V_n(A)$.

Inductive Step:

Assume that for all $j < n$, $V_n(B) \subseteq V_n(A)$.

$v \in V_n(B) \rightarrow v \in B_j \circ V_{n-j}(B) \rightarrow v \in B_j \circ V_{n-j}(A)$ because of induction hypothesis since $(n-j) < n$, $V_{n-j}(B) \subseteq V_{n-j}(A)$ for $j > 0 \rightarrow v \in A_j \circ V_{n-j}(A)$ since by lemma 5.8, $B_j = A_j$ for all possible B_j of B.

$\rightarrow V_n(B) \subseteq V_n(A)$. □

Lemma 5.10: Let A and B be generators for two distinct LRNs such that B is a prefix of A. Then $G_n(B) \subseteq G_n(A)$.

Proof:

$G_n(B) = (V_n(B), E_n(B))$

B is a prefix of A → By lemma 5.9, $V_n(B) \subseteq V_n(A) \rightarrow$ all the nodes in $G_n(B)$ are in $G_n(A)$.

Also $E_n(B) \subseteq E_n(A)$ since we are assuming that all the two graphs interconnect their identically.

$\rightarrow G_n(A) \subseteq G_n(B)$. □

Lemma 5.11: Let A and B be generators for two distinct LRNs such that B is a prefix of A. Let C be a generator such that $C \leq B$. Then $V_n(C) \subseteq V_n(A)$

Proof:

$C \leq B \rightarrow V_n(C) \subseteq V_n(B)$ by lemma 5.6.

B is a prefix of $A \rightarrow V_n(B) \subseteq V_n(A)$ by lemma 5.9.

$\rightarrow V_n(C) \subseteq V_n(B) \subseteq V_n(A)$

$\rightarrow V_n(C) \subseteq V_n(A)$. □

Lemma 5.12: Let A and B be generators for two distinct LRNs such that B is a prefix of A. Let C be a generator such that $C < B$. Then $G_n(C) \subseteq G_n(A)$

Proof:

Follows automatically from the fact that $G_n(C) \subseteq G_n(B)$ by lemma 5.6, and $G_n(B) \subseteq G_n(A)$ by lemma 5.9.

$\rightarrow G_n(C) \subseteq G_n(A)$. □

Figure 27 show the graphs of LRNs generated by A = 2.3, B = 2.1, and C = 2. Note that $G_n(C) \subseteq G_n(B)$ $G_n(A)$. We will omit the calculations here. They are similar to the calculations performed in the previous example.

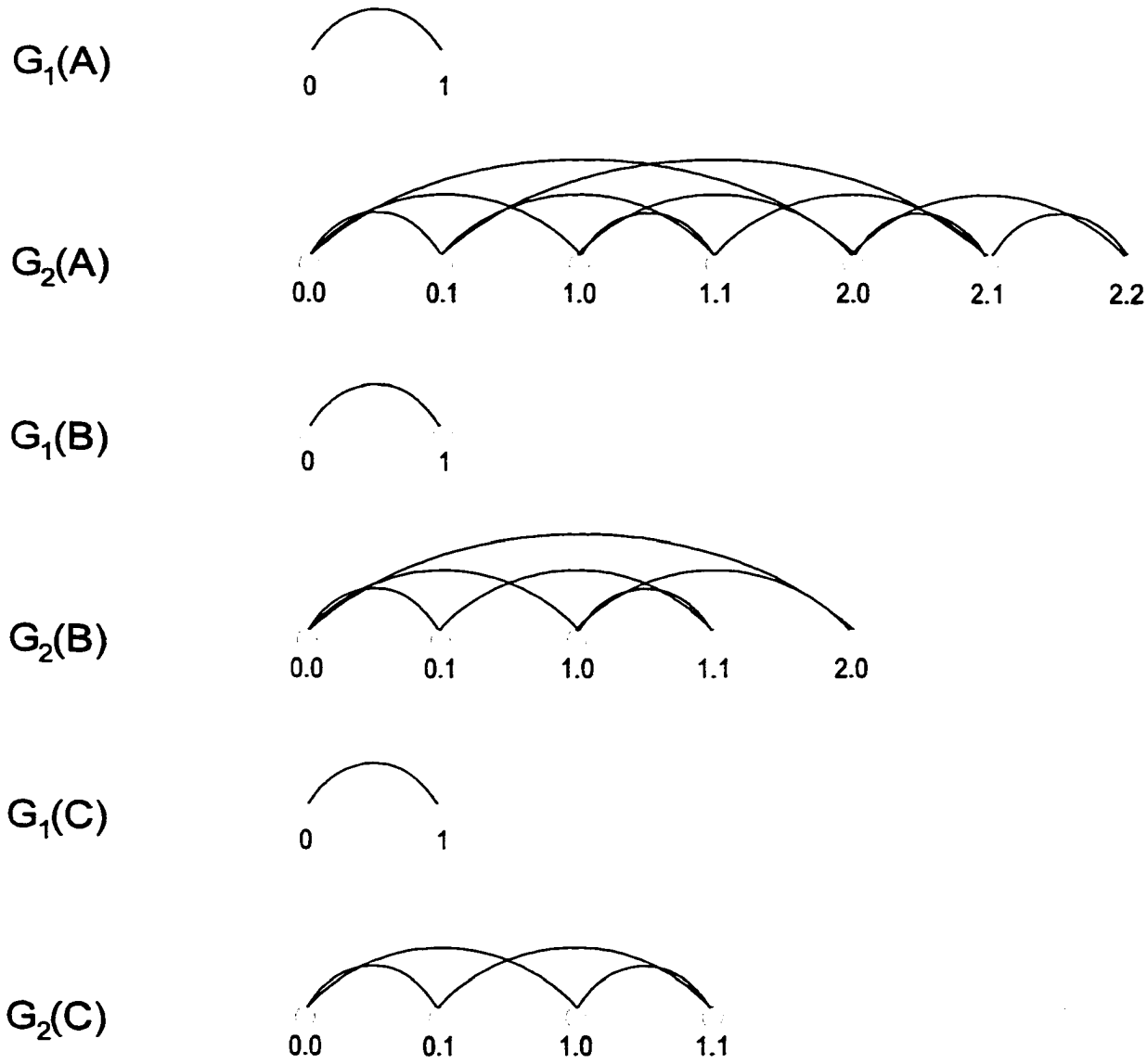


Figure 27: $G_n(C = 2) \subseteq G_n(B = 2.1) \subseteq G_n(A = 2.3)$.

Lemma 5.13: The n-cube can be embedded in any LRN generated by A = 2.s where s is a sequence of base-r numerals.

Proof:

The n-cube is generated by B = 2.

Since B ≤ A for all s, then by lemma 5.7, $G_n(B) \subseteq G_n(A)$ and therefore, the n-cube can be embedded in any LRN generated by A = 2.s. for all s. □

Lemma 5.14: Let A = 2*m where m ≥ 1. Therefore, $G_n(A)$ contains m^n n-cubes

Proof:

A =2*m is the generator of a LRN whose graph $G_n(A) = (V_n(A), E_n(A))$ where the size of the LRN is modeled by $X_n = 2*m\ X_{n-1}$. The solution to this recurrence equation is $(2m)^n$ meaning that there are $(2m)^n$ nodes in $G_n(A) \rightarrow |Vn(A)| = (2m)^n$.

An n-cube is modeled by B = 2, thus $X_n = 2\ X_{n-1}$ and therefore, there are 2^n nodes in $G_n(B) \rightarrow |V_n(B)| = 2^n$.

$\rightarrow |Vn(A)| = (2m)^n = 2^n * m^n = m^n * 2^n = m^n * |V_n(B)|$.

That is the LRN generated by A contains m^n copies of $|V_n(B)|$.

And Thus $G_n(A)$ contains m^n n-cubes. □

Lemma 5.15: Let A = 2*m, where m ≥ 1. Then $G_n(A)$ embeds at least m^n n-dimensional rings.

Proof:

Since it was shown in Lemma 5.14 that $G_n(A)$ contains m^n n-cubes, then $G_n(A)$ contains at least m^n n-dimensional rings since each n-cube embeds an n-dimensional ring. □

Lemma 5.16: Let A = 2*m where m ≥ 1 and m is even. Therefore, $G_n(A)$ contains $m^n/2$ (n+1)-cubes.

Proof:

An n-cube is modeled by B = 2, thus $X_n = 2\ X_{n-1}$ and therefore, there are 2^n nodes in $G_n(B) \rightarrow |V_n(B)| = 2^n \rightarrow |V_{n+1}(B)| = 2^{n+1}$.

By Lemma 5.14, the LRN generated by A contains m^n copies of an n-cube.

$\rightarrow |Vn(A)| = (2m)^n = m^n * 2^n = m^n/2 * 2 * 2^n = m^n/2 * 2^{n+1} = m^n/2 * |V_{n+1}(B)|$.

That is the LRN generated by A contains $m^n/2$ copies of $|V_{n+1}(B)|$.

And Thus $G_n(A)$ contains $m^n/2$ (n+1)-cubes. □

Lemma 5.17: Let A = 2*m, where m ≥ 1 and m is even. Then $G_n(A)$ embeds at least $m^n/2$ (n+1)-dimensional rings.

Proof:

Since it was shown in Lemma 5.16 that $G_n(A)$ contains $m^n/2$ (n+1)-cubes, then $G_n(A)$ contains at least $m^n/2$ (n+1)-dimensional rings since each n-cube embeds an n-dimensional ring, each (n+1)-cube embeds an (n+1)-dimensional ring. □

Lemma 5.18: Let A = 2*m where m ≥ 1 and m is an exact power of 2 → $m = 2^p$ where p ≥ 1. Therefore, $G_n(A)$ contains an (np+n)-cube, or (n(p+1))-cube.

Proof:

An n-cube is modeled by B = 2, thus $X_n = 2\ X_{n-1}$ and therefore, there are 2^n nodes in $G_n(B) \rightarrow |V_n(B)| = 2^n \rightarrow |V_{n(p+1)}(B)| = 2^{n(p+1)}$.

By Lemma 5.14, the LRN generated by A contains m^n copies of an n-cube.

$\rightarrow |Vn(A)| = (2m)^n = m^n * 2^n = (2^p)^n * 2^n = 2^{np} * 2^n = 2^{np+n} = 2^{n(p+1)} = |V_{n(p+1)}(B)|$.

Thus, $G_n(A)$ contains an (n(p+1))-cube. □

Lemma 5.19: Let A = 2*m where m ≥ 1 and m is an exact power of 2 → $m = 2^p$ where p ≥ 1. Then $G_n(A)$ is Hamiltonian.

Proof:

Since A = 2*m, where m ≥ 1, $m = 2^p$, and p ≥ 1. Then by Lemma 5.16, $G_n(A)$ contains a (n(p+1))-cube.

Let B = 2 be the generator of the (n(p+1))-cube $\rightarrow X_{n(p+1)}(B) = 2^{n(p+1)}$.

$X_n(A) = 2*m * X_{n-1}(A) = 2*2^p * X_{n-1}(A) = 2^{p+1} * X_{n-1}(A) = (2^{p+1})^n = 2^{n(p+1)} = X_{n(p+1)}(B)$.

So there are exactly the same number of vertices in $G_n(A)$ and $G_{n(p+1)}(B)$.

Since the n-cube is Hamiltonian $\rightarrow G_{n(p+1)}(B)$ is Hamiltonian $\rightarrow G_n(A)$ is Hamiltonian.

Thus if a LRN is generated by A = Since A = 2*m, where $m \geq 1$ and $m = 2^p$ where $p \geq 1$, then $G_n(A)$ is Hamiltonian. □

Lemma 5.20: Let A = 2*m where $m \geq 1$ and m is an exact power of $2 \rightarrow m = 2^p$ where $p \geq 1$. Then $G_n(A)$ can embed an n(p+1)-dimensional ring.

Proof:

The n(p+1)-dimensional ring is the Hamiltonian path of $G_n(A)$. □

Lemma 5.21: Let A = 1.2. Then $G_n(A)$ contains n/2-cube for all even n.

Proof:

$A = 1.2 \rightarrow A_1 = \{0\}$ and $A_2 = \{1.0, 1.1\}$

I define a mapping R: R(1.0) = 0, R(1.1) = 1.

If n = 0, then lemma 5.21 is vacuously true. $G_0(A) = \{\}$ and thus embeds a 0-cube whose graph is equal to {}.

Base case: n=2

$V_0(A) = \{\}$

$V_1(A) = \{0\}$

$V_2(A) = \{0.0, 1.0, 1.1\}$

$E_2(A) = \{(0.0, 1.0), (1.0, 1.1)\}$

Let $G' = (V', E') \subseteq G$ such that $V' = \{1.0, 1.1\}$ and $E' = \{(1.0, 1.1)\}$

Then G' is the graph of a 1-cube.

We can easily see that by applying R in the case $V'=\{0, 1\}$ and $E'=\{(0, 1)\}$, and conclude that G' is the graph of a 1-cube.

Inductive step:

We assume here that for any even integer n, G' is a subgraph of $G_n(A)$ and is the graph of a n/2-cube (after applying R to the label of the vertices). Let $G' = (V', G')$ such that $V' = \{l_1, \ldots l_m\}$

$G_{n+2}(A) = A_1 \circ G_{n+1}(A) + A_2 G_n(A)$

$G''=(V'', G'') \subseteq G_{n+2}(A)$ where $V'' \subseteq A_2 \circ Gn(A)$: $V'' = A_2 \circ V' \subseteq V_{n+2}$

$V'' = 1.0 \circ \{l_1, \ldots l_m\} \cup 1.1 \circ \{l_1, \ldots l_m\} = \{1.0.l_1, \ldots 1.0.l_m, 1.1.l_1, \ldots 1.1.l_m\}$

Note that $|V''| = 2 * |V'|$ where $|V'| = 2^{n/2} \rightarrow |V''| = 2^{n/2+1}$. $2^{n/2+1}$ is the number of nodes in a (n/2+1)-cube or and ((n+2)/2)-cube.

Each node label v of V'' can be mapped to R(v) to obtain: $V'' = \{0.l_1,\ldots,0.l_m1.l_1, \ldots 1.l_m\}$.

Since l_i represented node labels in an n/2 cube, than the labels in V'' will represent the node labels in an ((n+2)/2)-cube.

Now, we have the correct labels and the correct number of nodes in an ((n+2)/2)-cube.

To conclude that G'' is the graph of an ((n+2)/2)-cube we need to demonstrate that every edge in G'' is an edge in $G_{n+2}(A)$.

Let $x = x_1.x_2\ldots x_n$, $y = y_1.\ y_2\ldots y_n \in V''$ and let $(x, y) \in E''$.

Note that $x_1 = 0$ or $x_1 = 1$.. The same applies to y.

Note that, $x = x_1l_i$ and $y = y_1l_j$

Finally, we need to prove that:

$(x, y) \in E'' \rightarrow H(x, y) = 1 \rightarrow H(R^{-1}(x), R^{-1}(y)) = 1$. Thus $(R^{-1}(x), R^{-1}(y)) \in E_{n+2}$. R^{-1} is the inverse of R: $R^{-1}(0) = 1.0$, and $R^{-1}(1) = 1.1$.

Note that R(r.s) = R(r).R(s).

If $x_1 = y_1$ then $H(x, y) = H(x_1l_i, y_1l_j) = H(x_1l_i, x_1l_j) = 1 \rightarrow H(l_1, l_2) = 1$ where $l_1, l_2 \in V'$ and V' is the vertex set of G' which is the graph of an n/2-cube embedded in G_n.

Therefore, vertices $R^{-1}(l_1), R^{-1}(l_2) \in V_n$ corresponding respectively to $l_1, l_2 \in V'$, are connected in G_n

Therefore, $H(R^{-1}(l_1), R^{-1}(l_2)) = 1 \rightarrow H(R^{-1}(x_1).R^{-1}(l_1), R^{-1}(x_1).R^{-1}(l_2)) = 1$

$\rightarrow H(R^{-1}(x_1).R^{-1}(l_1), R^{-1}(y_1).R^{-1}(l_2)) = 1 \rightarrow (R^{-1}(x), R^{-1}(y)) \in E_{n+2}$.

If $x_1 \neq y_1$ then $H(x, y) = H(x_1l_i, y_1l_j) = H(x_1l_i, x_1l_j) = 1 \rightarrow H(l_1, l_2) = 0 \rightarrow l_1 = l_2$.

$\rightarrow R^{-1}(l_1) = R^{-1}(l_2) \rightarrow H(R^{-1}(l_1), R^{-1}(l_2)) = 0 \rightarrow H(R^{-1}(x_1).R^{-1}(l_1), R^{-1}(y_1).R^{-1}(l_2)) = 1$

$\rightarrow (R^{-1}(x), R^{-1}(y)) \in E_{n+2}$.

Therefore, the mapping R permits us to map a subset of $G_{n+2}(A)$ to an (n/2+1)-cube. □

Lemma 5.21 can be generalized as follows:

Lemma 5.22: Let A = 1.2. Then $G_n(A)$ contains n/2-cube for all $n \geq 0$.

Proof:

If n is even, then lemma 5.22 is true by lemma 5.21.

Otherwise, Note that $A_1 \circ V_{n-1}(A) \subseteq V_n(A)$ such that n-1 is even and thus by lemma 5.21 $V_{n-1}(A)$ embeds an ((n-1)/2)-cube. Therefore, $V_n(A)$ also embeds the same cube since the labels of that cube say $\{l_1, l_2, \ldots l_m\}$ now look like $\{0.l_1, 0.l_2, \ldots 0.l_m\}$ and this set represents a valid cube since the zero does not change anything in the hamming distance.

So $V_n(A)$ embeds an ((n-1)/2)-cube.

Since n is odd, n/2 = (n-1)/2 and thus $V_n(A)$ embed an (n/2)-cube. □

Lemma 5.23: Let A = s.2 where s is string of any base-r numerals. $G_n(A)$ contains an n/k-cube where k = |A| and n is multiple of k.

Proof:

The proof is similar to the proof of lemma 5.21. Define R(s.0) = 0, R(s, 1) = 1.

Lemma 5.24: Let A = s.2 where s is string of any base-r numerals. $G_n(A)$ contains an n/k-cube where k = |A| for all $n \geq 0$

Proof:

If n is a multiple of k, then lemma 5.24 is true by lemma 5.23.

Otherwise, note that $A_j \circ V_{n-j}(A) \subseteq V_n(A)$ such that n-j is a multiple of k and thus by lemma 5.23 $V_{n-j}(A)$ embeds an ((n-j)/k)-cube. Therefore, $V_n(A)$ also embeds the same cube since the labels of that cube say $\{l_1, l_2, \ldots l_m\}$ now look like $\{b.l_1, b.l_2, \ldots b.l_m\}$ where $b = a_1..a_{j-1}$, and this set represents a valid cube since the same concatenated sequence, b, does not change anything in the hamming distance.

So $V_n(A)$ embeds an ((n-j)/k)-cube. $(1 \leq j \leq k)$

Since n is not a multiple of k, n/k = (n-j)/k and thus $V_n(A)$ embed an (n/k)-cube. □

3 Partitioning in LRN

It is important to be able to partition networks. On each partition can run a job so to allow parallelism in the system. This way, the system will present a better overall performance. Also, in case of failures, it is helpful to think of the system as being a collection of partition so that if the failure occurs in a specific partition, the rest of the partitions can function correctly without bringing the system to a complete halt. This last issue is related to the fault tolerance of networks.

Lemma 5.25: LRNs can be partitioned to the same kind of networks of lower dimension. The partitions are determined by the generator or by an expansion of the generator.

Proof:

Follows automatically from the definition of LRNs and from Definition 5.4.

By definition, the generator of the LRN defines how the network of dimension n is composed in terms of lower dimensions, so that if $A = f_1(n).f_2(n)...f_k(n)$, the LRN is modeled by the recurrence equation $X_n = f_1(n)X_{n-1} + f_1(n)X_{n-1} + ... + f_k(n)X_{n-k}$ meaning that the graph $G_n(A)$ is made of $f_1(n)$ copies of $G_{n-2}(A)$, $f_2(n)$ copies of $G_{n-2}(A)$, ..., $f_k(n)$ copies of $G_{n-k}(A)$, and thus the partitionability of the LRN is automatic.

Otherwise, we can obtain a different way to partition a LRN generated by A, by finding expansions of A as defined in Definition 5.4. Every expansion of A gives a different way to partition the network in terms of lower dimensions. For example Let A = 2.5.8. This

tells us that we can partition the network, or $G_n(A)$ into two copies of $G_{n-1}(A)$, five copies of $G_{n-2}(A)$ and eight copies of $G_{n-3}(A)$. Let B = 1.7.13.8. B is an expansion of A. B dictates that the graph of the LRN generated by B therefore, by $G_n(A)$ can be partitioned into one copy of $G_{n-1}(A)$, seven copies of $G_{n-2}(A)$, thirteen copies of $G_{n-3}(A)$, and eight copies of $G_{n-3}(A)$. Each expansion of A gives a different way to partition the graph of $G_n(A)$. □

Lemma 5.26: Let A = 2*m where $m \geq 1$. Then $G_n(A)$ can be partitioned to m^n n-cubes

Proof:

Follows directly from Lemma 5.14.

$G_n(A)$ is made of m^n n-cubes and therefore, it can be partitioned to m^n n-cubes. □

Lemma 5.27: Let A = 2*m where $m \geq 1$. Then $G_n(A)$ can be partitioned to m^n n-dimensional rings

Proof:

Follows directly from Lemma 5.15.

$G_n(A)$ is made of m^n n-dimensional rings and therefore, it can be partitioned to m^n n-dimensional rings. □

Lemma 5.28: Let A = 2*m where m ≥ 1 and m is even. Then $G_n(A)$ can be partitioned to $m^n/2$ (n+1)-cubes.

Proof:

Follows directly from Lemma 5.16.

$G_n(A)$ is made of $m^n/2$ (n+1)-cubes and therefore it can be partitioned to $m^n/2$ (n+1)-cubes. □

Lemma 5.29: Let A = 2*m where m ≥ 1 and m is even. Then $G_n(A)$ can be partitioned to $m^n/2$ (n+1)-dimensional rings.

Proof:

Follows directly from Lemma 5.17.

$G_n(A)$ is made of $m^n/2$ (n+1)-dimensional rings and therefore it can be partitioned to $m^n/2$ (n+1)- dimensional rings. □

Lemma 5.30: Let A = 2*m where m ≥ 1 and m is an exact power of 2 → $m = 2^p$ where p ≥ 1. Therefore, $G_n(A)$ partitioned to an (np+n)-cube, or (n(p+1))-cube.

Proof:

Follows directly from Lemma 5.18.

$G_n(A)$ is made of an (np+n)-cube and therefore it can be partitioned an (np+n)-cube. □

Lemma 5.31: Let A = 2*m where m ≥ 1 and m is an exact power of 2 → $m = 2^p$ where p ≥ 1. Therefore, $G_n(A)$ contains an (np+n)-dimensional ring, or (n(p+1))-dimensional ring.

Proof: n(p+1)-dimensional ring

Follows directly from Lemma 5.20.

$G_n(A)$ is made of an (np+n)-dimensional ring and therefore it can be partitioned an (np+n)-dimensional ring. □

Chapter 6

The Fastcube:

A Variation on Hypercube Topology with Lower Diameter

The class of n-cubes and its variance are generally considered to be one of the most popular classes of interconnection networks. The popularity of n-cubes is due to their small diameter, ease of routing and the rich embedding properties. In addition, a large number of efficient parallel algorithms have been developed for n-cubes.

A number of approaches are suggested to increase the performance of an n-cube by reducing its diameter. Twisted Hypercube reported in [11, 1] is an example of such a topology. In a twisted 3-cube the endpoints of two edges are exchanged. As a result, the diameter of the network is reduced from three to two. This is the minimum diameter that can be obtained without changing the resources of the network, such as nodes and links. Further reduction of the diameter of the 3-cube requires sixteen edges.

In recent years, several new cube-like topologies have been presented [11, 22, 1, 26]. These topologies have lower diameter than a same size cube. The primary disadvantage of these new topologies is their complex routing and embedding algorithms.

This chapter presents a new cube-like interconnection topology, a class of LRN with $N = 2^n$ nodes which we refer to as *n-fastcubes*. This class of interconnection networks is a class of LRNs generated by $A = 2$. The fastcube has smaller diameter and mean

internodal distance than the same size cube. The node degree of the n-fastcube is n and its diameter is $\lceil (n+1)/2 \rceil$, which is substantially smaller than that of the same size Hypercube. Topological properties, as well as several routing algorithms, are developed for fastcubes. In addition, a new methodology for the design and analysis of fastcubes is employed. This methodology is based on modeling interconnection networks as finite state automata. The inputs to these particular automata are routing sequences. The routing and embedding algorithms developed in this chapter produce routing sequences. The main characteristic of routing sequences is their node independence.

There are two approaches to reducing the diameter of an interconnection network. The first approach is to introduce additional links while keeping the underlying network topology intact. This approach has resulted in many variations of enhanced topologies such as enhanced Hypercubes and enhanced rings. The main disadvantage of this approach is the increase cost of implementation and more complex routing and embedding algorithms.

The second approach, which is first reported in [15, 18] and employed in this chapter, is to increase the number of routing functions and selectively apply a subset of these functions to each node. For instance, if an n-bit string encodes each node label of the network, each bit of the string is used as the key to select the routing function, which is applicable to that node.

In the past, a large number of research activities toward reducing the diameter of Hypercubes have been conducted. In [11], Estafahanian et. Al. have proposed a technique to generalize the Twisted 3-cube by increasing its dimension to N. The Twisted N-cube

has diameter of one less than of the Hypercube of the same dimension. The runtime of the routing algorithm for Twisted N-cube is the same as Hypercube and is O(n).

In [22], Hillbers. et. Al., have proposed an n-dimensional Twisted cube constructed by linking four twisted cubes of dimension n-2. The links connecting the four n-2 twisted cubes are also twisted so that these twisted cubes are in a Hamming distance of one from each other. The diameter of the resulting Twisted cube is $\lceil (n+1)/2 \rceil$.

The routing algorithms developed for Twisted cubes are significantly more complex than the Hypercubes.

A new topology similar to the Twisted cube is introduced in [10]. This new topology called Multiply Twisted cube, joins four (n-2) cubes together to form an n-cube. The structure of Multiply Twisted cube differs from the Twisted cube in that its connection rules results in communication links to be twisted concurrently across several dimensions. The diameter of the n-dimensional Multiply Twisted cube is also $\lceil (n+1)/2 \rceil$. Several point-to-point and broadcast routing have been developed for Multiply Twisted cubes and reported in [10].

The most recent improved cube type topology is Mobius cubes, which is introduced in [7]. The Mobius cubes also generalize the Twisted 3-cube in expense of more complex routing and decomposition algorithms.

1 Notations

This section presents notations that will be used throughout this chapter.

B denotes the set containing two elements 0 and 1.

$\{B\}^n$ is the n-dimensional vector space defined over B.

A vector $X=(x_{n-1} \ldots x_0) \in \{B\}^n$ is a sequence of n bits, where x_i is the component of X along dimension i of the vector space $\{B\}^n$.

For a vector $X=(x_{n-1} \ldots x_0) \in \{B\}^n$, I(X) is the set of all indices i for which x_i is non-zero. For example, I(X)={5,3,2} for X=(0101100).

X^c is the vector obtained from a vector X by complementing all components of X.

#(0) and #(1) denote the number of zeros and ones, respectively, in a vector X.

$e_i \in \{B\}^n$ is the vector with only its ith dimension equal 1.

E_i , $1 \leq i \leq n-1$ is the vector with one in positions i and (i-1) and zero otherwise. For example for n = 4, $e_2 = 0100$ and $E_2 = 0110$.

+ denotes modulo-2 addition.

X+Y is the vector obtained by bit-wised mod-2 addition of components of vectors X and Y.

For $a \in B$ and $X \in \{B\}^n$, aX denotes the vector obtained by multiplying (ANDing) each component of X by a.

R denotes a redundant base for vector space $\{B\}^n$.

Expansion of a vector X in $\{B\}^n$ by R is denoted by E(X), i.e., E(X) is a set of basis vectors in R which expands X.

W(X) denotes the minimum weight of the expansion of X, i.e., W(X) is the cardinality of the E(X) with minimum number of components.

For example, let X=(1010), Y=(1100) and a=1. Then, X^c=(0101), X+Y=(1110), and aX=X.

2 Redundant Basis and Minimal Expansions

This section presents several algorithms for the expansion of a vector in $\{B\}^n$. The expansion techniques are fundamental to our future discussion on routing and embedding properties of the proposed class of interconnection networks.

Definition 6.1: *Let $P=\{p_i \in \{B\}^n,\ 0 \le i \le n-1\}$. P forms a non-redundant basis for $\{B\}^n$ if and only if:*

For any vector $X \in \{B\}^n$, X can be decomposed as a linear sum of basis vectors, i.e., $X = \Sigma_i a_i.p_i$ *(1)*

p_i's, $\forall i$, are linearly independent, i.e., for $X=p_i$ the only nonzero coefficient of the expansion (1) is $a_i=1$.

For example, the set $\{e_i:\ 0 \le i \le n-1\}$ form a non-redundant basis for vector space $\{B\}^n$.

Definition 6.2: *A redundant set of basis for the vector space $\{B\}^n$ with 2n elements is denoted by $R=\{p_i, q_i: 0 \le i \le n-1\}$.*

Any vector X in $\{B\}^n$ can be expanded by R is in the form:

$X = \Sigma_i (a_i p_i + b_i q_i)$ *with* $a_i, b_i \in B$ *and* $0 \le i \le n-1$. (2)

For example, the set $\{e_i , e_i^c : 0 \le i \le n-1\}$ forms a redundant basis for $\{B\}^n$.

Definition 6.3: *The set E(X) of p_i and q_i with non-zero coefficients in (2) is said to be an expansion of X. The cardinality of E is called the weight of the expansion.*

Definition 6.4: *If a vector $X \in \{B\}^n$ has several expansions by a redundant basis, the expansion with minimal weight is referred to as minimal expansion of X and the corresponding weight is denoted by W(X).*

In the remainder of this chapter, since we are interested in the minimal expansion of a vector X by a redundant set R, we use E(X) and W(X) to denote the minimal expansion set and minimal weight of the expansion respectively.

2.1 Properties of Minimal Expansions

The minimal expansion of a vector has several properties. Some of these properties are explored in this section.

Theorem 6.1: *The following sets of redundant basis span any vector X in the vector space* $\{B\}^n$ *with* $W(X) \leq \lceil n/2 \rceil$

$R=\{e_i, E_i: 0 \leq i \leq n-2\}$

$R=\{e_i, e^c_i: 0 \leq i \leq n-1\}$

Proof: We prove this theorem by developing two algorithms, which expand a non-trivial vector X with $W(X) \leq \lceil n/2 \rceil$. A non-trivial vector has a label, which contains both 0 and 1 digits. The zero and identity vectors are trivial and their expansions are described in the next theorem.

Algorithm 1:

(An expansion of X based on the redundant set of part a)

Input: An n-vector X,

Output: A minimal expansion of X with $W(X) \leq \lceil n/2 \rceil$

```
E(X) = N        /*set the expansion set E to null*/
I = n - 1
While (I > 0)
Begin
```

Scan the vector X from left to right starting from bit position i and find the first 1 in the sequence. Assume that the 1 appears in position j.

If (j == 0) exit loop

If (X_{j-1} == 0)

E(X) = E(X)U{e_j}

Else

E(X) = E(X)U{E_j}

I = j - 2

End

If (I == 0 and X_I == 1) */*At this point I ≤ 0*/*

E(X) = E(X)U{e_j}

Output E(X).

Algorithm-1 decomposes any non-zero vector. The case of $X=0^n$ (0^n denotes n times concatenation of 0) will be discussed in the Lemma 6.1. As to $X=1^n$, the algorithm will output a minimum expansion $\leq \lceil n/2 \rceil$

Algorithm 2: *(An expansion of X based on the redundant set of part b)*

Input: An n-vector X,

Output: A minimal expansion of X with W(X) ≤ $\lceil n/2 \rceil$

Step 1) E(X)=N /set the expansion set E to null/

Step 2) if min(#0, #1)= #1 do /clearly min(#0,#1) ≤⌈n/2⌉/

while I(X) is non-empty do

$E(X):=E(X)U\{e_i\}$, $i\ 0\ I(X)$

$I(X) := I(X) - \{i\}$

enddo

Step 3) if min(#0, #1) = #0 do

while $I^c(X)$ is non-empty do /$I^c(X)$ is the complement of the set I(X), i.e., the set of all indices of zeros in the vector X./

$E(X):=E(X)\ U\{e^c_i\}$, $i\ 0\ I^c(X)$

$I^c(X):=I^c(X) - \{i\}$

enddo

output E(X).

Lemma 6.1: The weight of the minimal expansions of zero and identity vectors, using redundant basis in part (a) of the Theorem 6.1, are 2 and $\lceil n/2 \rceil$ respectively.

Proof:

Clearly, $0^n = e_i + e_i$,$\forall\ 0 \le i \le n\text{-}1$. $E(0^n) = \{e_i, e_i\}$ and therefore $W(0^n) = 2$.

If we input $X = 1^n$ to algorithm 1, we will get $E(X) = \{E_{n-1}, E_{n-3}, \ldots, E_1\}$ if n is even or $E(X) = \{E_{n-1}, E_{n-3}, \ldots, E_2, e_0\}$ if n is odd. Algorithm1 produces an E_i for every two consecutive ones.

If n is even, we will get exactly n/2 groups i.e. n/2 E_I and thus $W(X) = \lceil n/2 \rceil$

If n is even, we will get exactly n/2 groups of E_I and e_0 to make a total of (n/2 + 1) groups and thus $W(X) = n/2 + 1 = \lceil n/2 \rceil$

Theorem 6.2: *If R is a minimal redundant basis for the vector space$\{B\}^n$, then $e_i \in R$ for all possible i.*

Proof: By contradiction. Consider the 3-dimensional vector space. To expand e_i at least two basis vectors are required, which implies that R is not minimal.

3 Modeling the Hypercube with a non-redundant basis

In this section, we will model the Hypercube by a set of non-redundant basis. We will later extend this technique to describe the Supercube and finally, n-fastcube topology. The proposed methodology will also be used to investigate the routing and embedding properties of fastcubes. It should however be noted that, the purpose of modeling the class of n-cubes is to present the proposed modeling technique and to demonstrate its application to the design and the analysis of communication networks.

Definition 6.5: *Let P_n and $P = \{e_i \in \{B\}^n,\ 0 \leq i \leq n-1\}$ denote the set of all binary sequences of length n and a non-redundant basis respectively. The transition function δ: $P_n \times P \rightarrow P_n$ is defined as $\delta(X, e_i)=X+e_i$ for all $X \in P_n$ and $0 \leq i \leq n-1$.*

For a node X, e_i defines the immediate neighbor of X along dimension i of an n-cube. For this reason, we refer to e_i as routing symbol.

A path between two nodes X and Y involves several intermediate nodes. To connect these nodes, a sequence of routing symbols should be applied to X. The next two definitions formalizes the concept of a routing sequence and assigns a unique characteristic to each routing sequence.

Definition 6.6: *A routing sequence p of length k is a concatenation of k routing symbols.*

Definition 6.7: *The transition function δ can be extended to cover routing sequences as follows: If p is a routing sequence with e_i as the first symbol, i.e., $p=e_ip'$, then $\delta(X,p)=\delta(\delta(X,e_i), p')$ for all routing sequences p and p' and for all routing symbols e_i.*

Definition 6.8: *The signature of a routing sequence p, denoted by S(p), is modulo-2 sum of all the routing symbols in p.*

For instance, the routing sequence $p=e_1e_3e_5$ in a 6-cube has signature of 101010.

Note that the sequence $e_1e_4e_3e_4e_5$ has the same signature as the sequence $e_1e_3e_5$. Thus, if a routing symbol appears several times in a routing sequence, that sequence can be shorten by deleting pairs of identical routing symbols.

Definition 6.9: *Two routing sequences p and p' are said to be equivalent, $p \sim p'$, if and only if $S(p)=S(p')$.*

The following theorem reveals some of the important properties of routing sequences.

Theorem 6.3: *Let p and p' be two routing sequences. Then,*

p and p' connects the same two nodes together if and only if $p \sim p'$.

The path established by the sequence p is minimal, i.e.; it can not be replaced by a shorter path connecting the same two nodes, if and only if p composed of distinct routing symbols. We refer to p as a minimal routing sequence.

If p is a minimal routing sequence and p and p' are equivalent, then p' is composed of all routing symbols of p, plus an even number of occurrence of identical routing symbols.

If H(X, Y) denotes the Hamming distance between nodes X and Y, then H(X, Y) is the signature of all routing sequences which are capable of connecting node X to node Y.

Proof:

Let (X, Y) be the source-destination pair of nodes.

Since p and p' connect X to Y, $\delta(X, p) = Y$ and $\delta(X, p') = Y$. Thus, $\delta(X, p) = \delta(X, p')$ which in turn implies that: $\delta(X, p) = X + p_1 + \ldots + p_n = \delta(X, p') = X + p'_1 + \ldots + p'_n$.

Thus, $p_1 + \ldots + p_n = p'_1 + \ldots + p'_n$ or $S(p) = S(p')$ or $p \sim p'$.

To show sufficiency, let $p \sim p'$. This implies that $S(p) = S(p')$ or $p_1 + \ldots + p_n = p'_1 + \ldots + p'_n$. Therefore, $X + p_1 + \ldots + p_n = X + p'_1 + \ldots + p'_n$ or $\delta(X, p) = \delta(X, p')$

if $\delta(X, p) = Y$, p connects X to Y. Also since $\delta(X, p') = \delta(X, p) = Y$, p' also connects X to Y.

Suppose that routing sequence p connecting nodes X to Y is not composed of distinct symbols. Let $p = e_1 \ldots e_x \ldots e_x \ldots e_n$, and suppose that the path established by p is minimal. $\delta(X, p) = Y$ implies that $X + e_1 + \ldots + e_x + \ldots + e_x + \ldots + e_n = Y$. But since the symbol e_x occurs twice, we can collapse it (in modulo-2 addition $x+x = 0$) and obtain $X + e_1 + \ldots + e_n = X + P' = Y$. Clearly p' also routes X to Y since $\delta(X, p') = Y$ as shown above. But $|p'| = |p| - 2 < |p|$, since two symbols have been eliminated from p. Thus, the path established by p is not minimal which contradicts the assumption.

To show sufficiency, suppose that p is composed of distinct symbols but is not a minimal length routing sequence. Therefore, there exists a routing sequence p' that connects X to Y such that $|p'| < |p|$ (1).

Since p and p' both connect X to Y, by (a), $p \sim p'$ implies that $S(p) = S(p')$. Since p is made up of distinct symbols and $S(p) = S(p')$, $|p'| >= |p|$: which is a contradiction with (1).

Since p ~ p', S(p) = S(p'). Furthermore, by assumption p is minimal. Thus, p is composed of distinct routing symbols. From the fact that |p'| >= |p|, we consider the following cases.

Case 1: |p'| =|p|.

Since S(p') = S(p), p' = p and therefore p' is made up of all symbols of p, plus an even number of occurrence of identical routing symbols.

Case2: |p'| >|p|.

In this case, either |p'| =|p| + 2n for n>0, or |p'| =|p| + (2n+1) for n>=0

if |p'| =|p| + 2n, the 2n additional symbols are either identical or non-identical,

If they are identical the additional symbols are going to pair off and cancel each other, and only those symbols from p will be left.

If they are not all identical the additional symbols, at least one pair of symbols will not cancel each other and p' will be made of all the elements of p plus 2 distinct elements → S(p') ≠ S(p'). So this can not be the case since we assumed S(p') = S(p)

if |p'| =|p| + 2n+1, then one can show, by contradiction, that S(p') ≠ S(p), which is a contradiction since p and p' are equivalent.

To establish a path form X to Y, H(X, Y) = X ⊕ Y is first calculated.

Let p be a routing sequence that connects X to Y, p = $\{e_1,\ldots,e_n\}$, S(p) =e_1 +...+ e_n. δ(X, p) = Y implies that X + e_1 + ... + e_n = Y, or X + e_1 + ... + e_n + Y = 0. Rearranging the terms results in X + Y = e_1 +...+ e_n = S(p) → H(X, Y) = X ⊕ Y = e_1 + e_n = S(p).

Therefore, if p connects X to Y, H(X, Y) = S(p).

We now present a formal topological description of an n-cube.

Definition 6.10: *An n-cube is an ensemble of 2^n nodes which are labeled from 0 to 2^n-1. Two nodes are directly connected if and only if their Hamming distance can be expanded by a single routing symbol.*

It is rather obvious that the set $\{e_i : 0 \leq i \leq n-1\}$ forms a non-redundant basis for an n-cube. Thus, any vector in $\{B\}^n$ can be decomposed into a sequence of routing symbols. This observation makes the routing in an n-cube rather straightforward. The following algorithm describes the routing methodology in an n-cube.

Algorithm 3: Cube Router

Input: X and Y, the source and destination node labels.

Output: A minimal routing sequence.

Form H(X, Y), the Hamming distance of X and Y.

2- Decompose H(X, Y) using the non-redundant basis $\{e_i : 0 \leq i \leq n-1\}$.

3- The minimal routing sequence p is obtained by concatenating all nonzero basis in the expansion of H(X, Y).

It should be noted that any permutation on the routing symbols of p forms a new sequence p' which has the same signature as p. Thus, p' is also a minimal routing sequence. Based on the above observation, one can conclude the following:

Lemma 6.2: The formulation of the n-cube as presented by the non-redundant basis model is correct.

We need to prove that:

The neighboring defined by the above model is equivalent to the $Cube_i$ neighboring function defined for the n-cube in the literature

The Graph generated by the above model is effectively a cube

Proof:

In the above model, two nodes are neighbors if and only if $\delta(v, e_i)$ for some node $v = v = v_{n-1}.v_{n-2}\ldots v_1v_0$.

$\delta(v, e_i) = v + e_i = v_{n-1}.v_{n-2}\ldots v_1v_0 + e_i = v_{n-1}.v_{n-2}\ldots v_i'\ldots v_1v_0 = Cube_i(v)$.

Therefore neighboring as defined by the above model is equivalent to $Cube_i$.

$G_n(A) = (V_n(A), E_n(A))$. $V_n(A)$ is the node set and is prescribed by the recursion since n-cube is a LRN. The above-described methodology does not interfere nor change the node set.

The edge set is dictated by the neighboring function. The neighboring function as defined by the above model is equivalent to the $Cube_i$. Therefore, the edge set is the edge of an n-cube and thus $G_n(A)$ is the graph of an n-cube.

Lemma 6.3: *Let #1 denotes the number of 1's in the Hamming sequence H(X, Y). Then, there exist (#1)! minimal length paths between X and Y.*

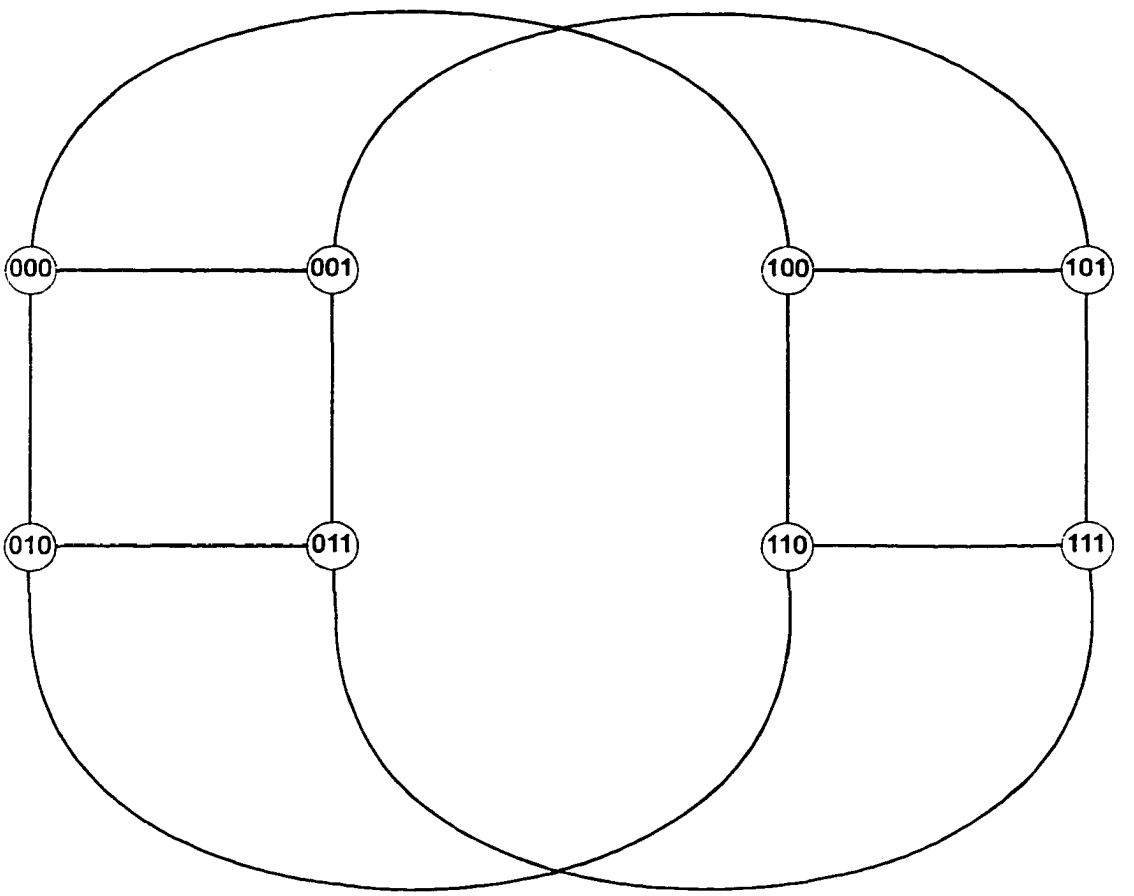


Figure 28: An 3-cube.

For instance, there are 3!=6 minimal length paths connecting node 010 to node 101. The corresponding routing sequences are: $e_1e_2e_3$, $e_1e_3e_2$, $e_2e_1e_3$, $e_2e_3e_1$, $e_3e_2e_1$, and $e_3e_1e_2$.

The neighbors of a node with label $a_{n-1}...a_i...a_0$ are $a_{n-1}...a_i...a_0 + e_i$, $0 \leq i \leq n-1$, i.e., the neighbors of a node $a_{n-1}...a_i...a_0$ are defined by modulo-2 addition of non-redundant basis, $\{e_i\}$, to each one of n-dimensions of the node label.

$\{E_i : 0 \leq i \leq n-1\}$ forms also a non-redundant basis for an n-cube. The n-cube can be modeled using this non-redundant basis where nodes X and Y are neighbors if and only if $X = E_i + Y$ or $Y = E_i + X$. The resulting graph is depicted in Figure 29.

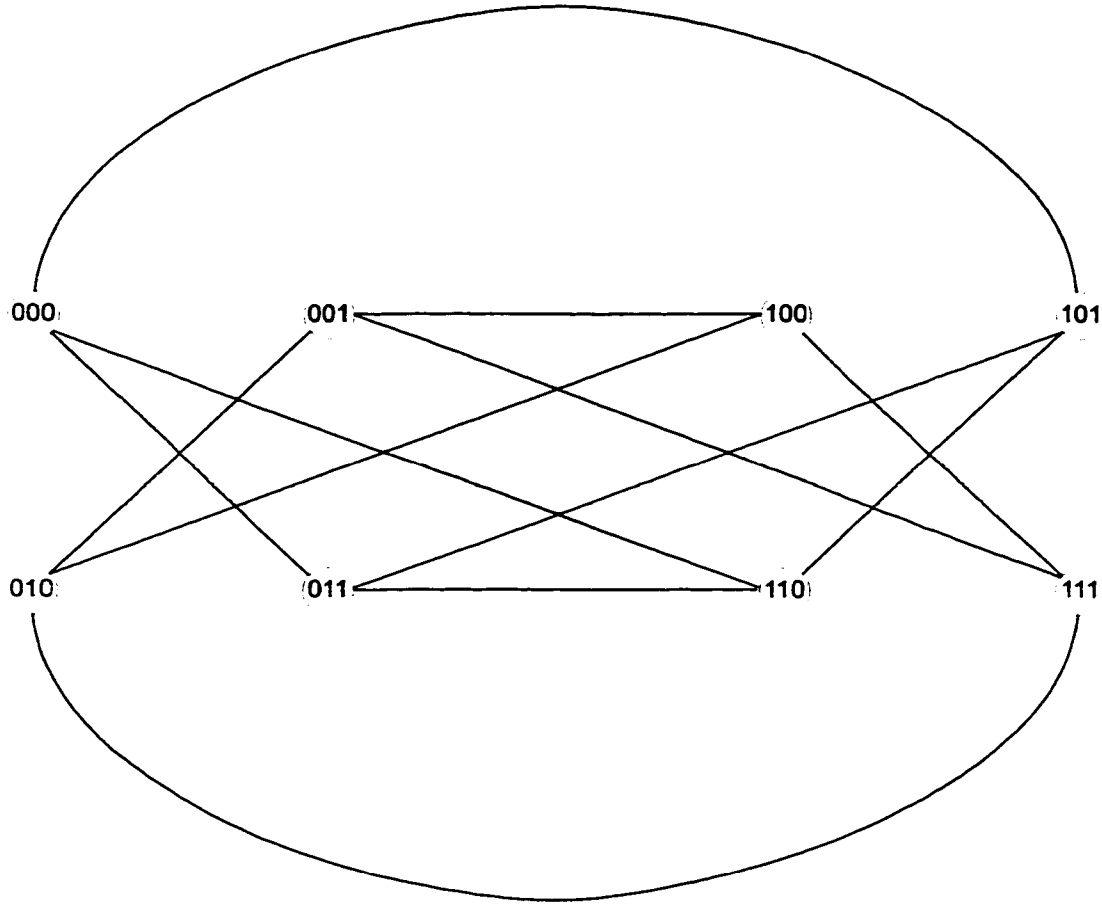


Figure 29: An isomorphic graph to a 3-cube.

We will introduce in the next section a topology called the Supercube. The graph of a Supercube can be obtained by superposing graphs from Figure 28 and Figure 29.

4 Supercube

In this section, we introduce an enhanced n-cube, which we call a Supercube.

A Supercube connects 2^n nodes and has diameter of $\lceil n/2 \rceil$ and node degree of 2n.

Definition 6.11: *Let $p=p_0 p_1 ... p_i ... p_{n-1}$ be a routing sequence of length n defined over the redundant set $R = \{e_i, E_i / 0 \leq i \leq n-1\}$, i.e., each routing symbol p_i is either e_i or E_i.*

p is said to be applicable to a node v if and only if the mapping $\delta(v, p_0 p_1 ... p_i)$ is defined for all i, $0 \leq i \leq n-1$.

For example, all routing sequences defined over the set $\{e_i, E_i / 0 \leq i \leq n-1\}$ are applicable to any node of a Supercube. That is because in each dimension of a Supercube both types of edges exist.

There are two types of link (edge) in a super cube, e-type or ***e-link***, and E-type or ***E-link***. (x, y) is a type e-link if and only if $x = y + e_i$ for some i, $0 \leq i \leq n-1$ and is of E-type if and only if $x = y + E_i$. The transition function δ of a Supercube is defined for each pair of (x, e_i) and (x, E_i). Thus, along each dimension i of a node, there are two links one of each types. Figure 30 shows the topology of the 3-Supercube.

To connect two nodes X and Y of a Supercube, a routing sequence p of length no greater than n/2 can be formed such that, $\delta(X, p) = Y$ and $\delta(Y, p) = X$. The routing sequence p depends on H(X, Y) and can be generated from Algorithm 4.

The structure of a SuperCube is such that any routing sequence p is applicable to any node in the network. In other words, $\delta(X, p)$ is defined for any pair (X, p).

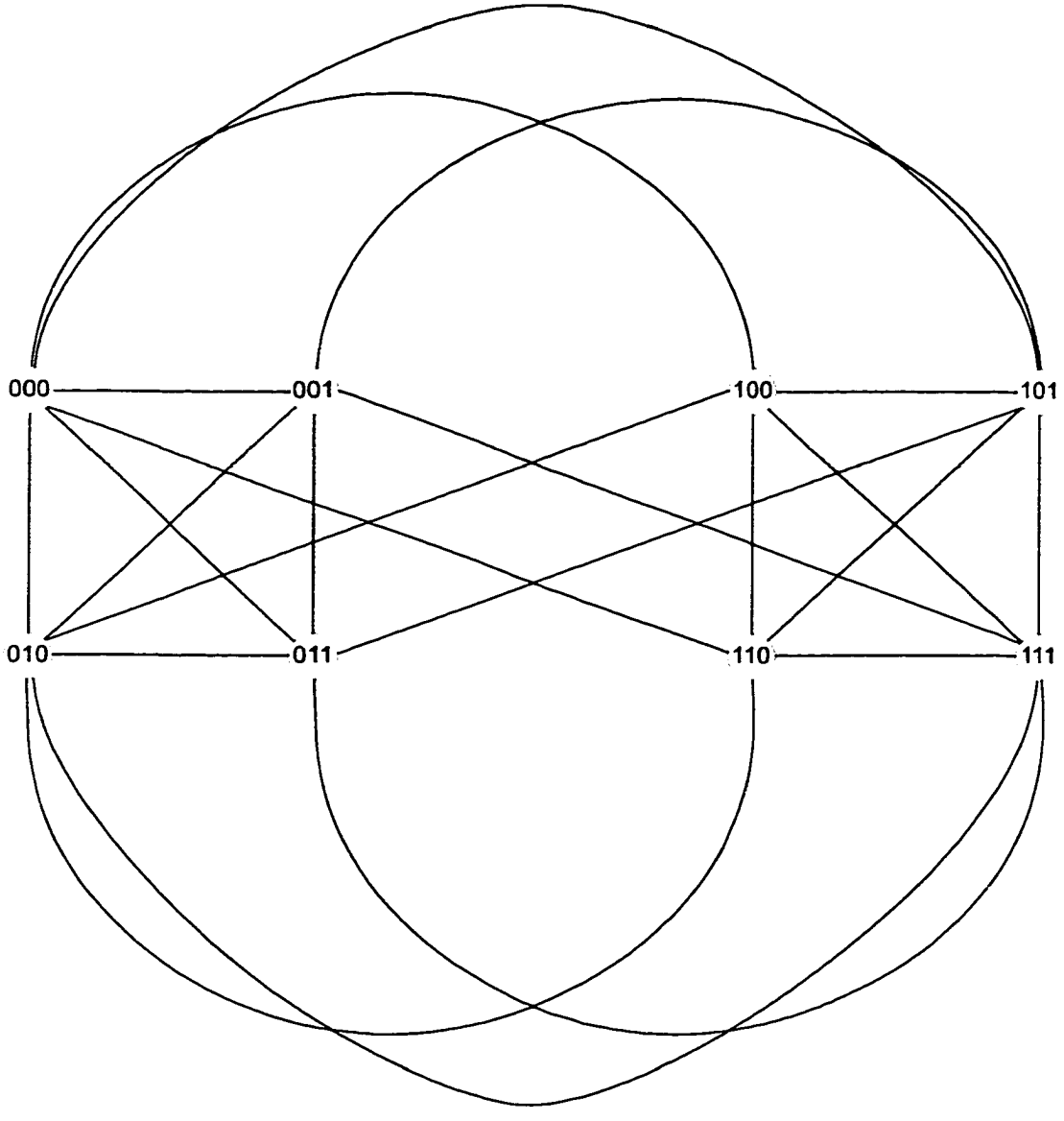


Figure 30: An 3-Supercube.

Definition 6.12: A Supercube is a LRN with generator A = 2. The graph of an Supercube $G_n(A) = (V_n(A), E_n(A))$ is super graph of the n-cube such that the vertex $V_n(A)$ is the same but the edge set $E_n(A)$ is defined as follows: $(v_1, v_2) \in E_n(A)$ if $v_1 = v_2 + e_i$, or $v_1 = v_2 + E_i$.

Since both e_i and E_i are applicable at every node of the Supercube, routing is very simple. It is very similar to routing in the n-cube using the algorithm3 except that the decomposition of the Hamming distance has to be in function of the redundant basis E = $\{e_i, E_i : 0 \leq i \leq n-1\}$ forms a redundant basis for the Supercube. Thus, any vector in $\{B\}^n$ can be decomposed into a sequence of routing symbols. This observation makes the routing in an Supercube rather straightforward. The following algorithm describes the routing methodology in an n-cube.

Algorithm 4.: SuperCube Router

Input: X and Y, the source and destination node labels.

Output: A minimal routing sequence.

Form H(X, Y), the Hamming distance of X and Y.

2- Decompose H(X, Y) using the non-redundant basis $\{e_i, E_i : 0 \leq i \leq n-1\}$.

3- The minimal routing sequence p is obtained by concatenating all nonzero basis in the expansion of H(X, Y).

Lemma 6.4: The diameter of the Supercube, D(n), is upper bounded by $\lceil n/2 \rceil$.

Proof:

Let X and Y $\in V_n(A)$ where A = 2 is the generator of the Supercube.

To connect X to Y we first have to find H(X, Y), decompose it, and use its non-zero components to route X to Y.

The length of maximum length of the decomposition is W(H(X, Y)).

The weight of H(X, Y) ≤ n since X and Y are made of n base-r numeral so that the maximum difference in the labels can not exceed n.

Therefore by Theorem 6.1 $W(H(X, Y)) \leq \lceil n/2 \rceil$

Therefore the number of non-zero components obtained from the decomposition of H(X, Y) $\leq \lceil n/2 \rceil$.

Therefore, The length of the path form X to Y can not exceed $\lceil n/2 \rceil$

Thus $D(n) \leq \lceil n/2 \rceil$. □

The Supercube is an enhanced cube that improves the diameter of cube by a factor of ½ , but the cost of the network goes up since many more links are introduced, so that the node degree of the Supercube is 2n as to n for the n-cube. In the next section, we will develop an interconnection network which graph is a sub-graph of the Supercube. Its cost is the same as the n-cube, but it presents a far better diameter than the cube, namely $\lceil (n+1)/2 \rceil$.

5 Fastcube Networks

The n-fastcube has the same number of nodes as the n-cube, and the Supercube. The graph of the n-fastcube is a subgraph of the Supercube. Its diameter is slightly bigger,

namely $\lceil (n+1)/2 \rceil$. The n-fastcube is an alternative to the cube but present a far better diameter without increasing the cost of the network i.e. number of nodes and number of edges.

Definition 6.13: A fastcube of dimension n is a LRN generated by A = 2 with a total of $N = 2^n$ nodes. Let $v \in V_n(A)$. The neighbors of v are: $\begin{cases} v + e_{i+1} \text{ if } v_{i+1} = 0 \\ v + E_i \text{ Otherwise.} \end{cases}$

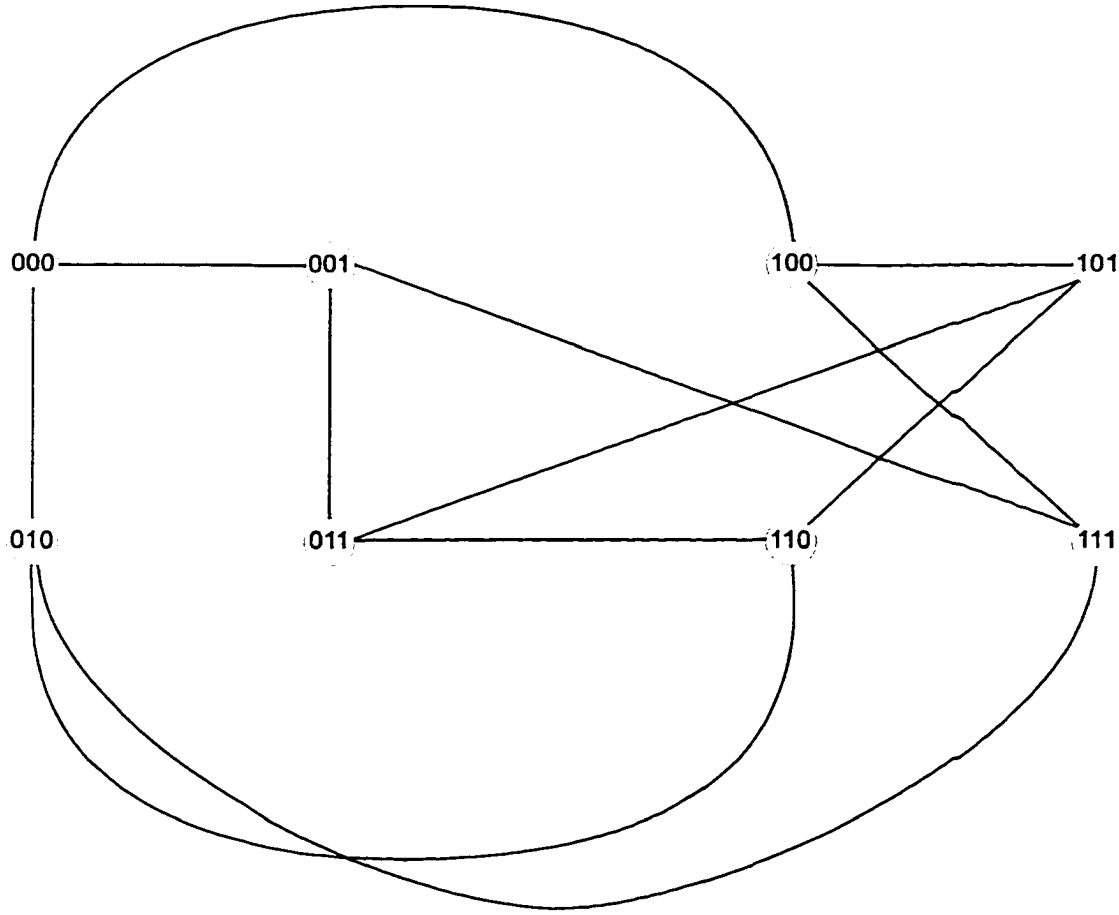


Figure 31: A 3-fastcube

Thus, along dimension i of a node v, there is a single link of type e or E, which is determine by the bit v_{i+1}. This dependency of the link type to the node labels make some routing sequences inapplicable to some nodes.

For instance, the sequence E_4e_2 is inapplicable to the node 01111. This is in contrast to the SuperCube where both links are defined at every node. The topology of the 3-fastcube is depicted in Figure 31. Figure 32 is equivalent to Figure 31 but it is a more elegant way to graphically show the n-fast cube.

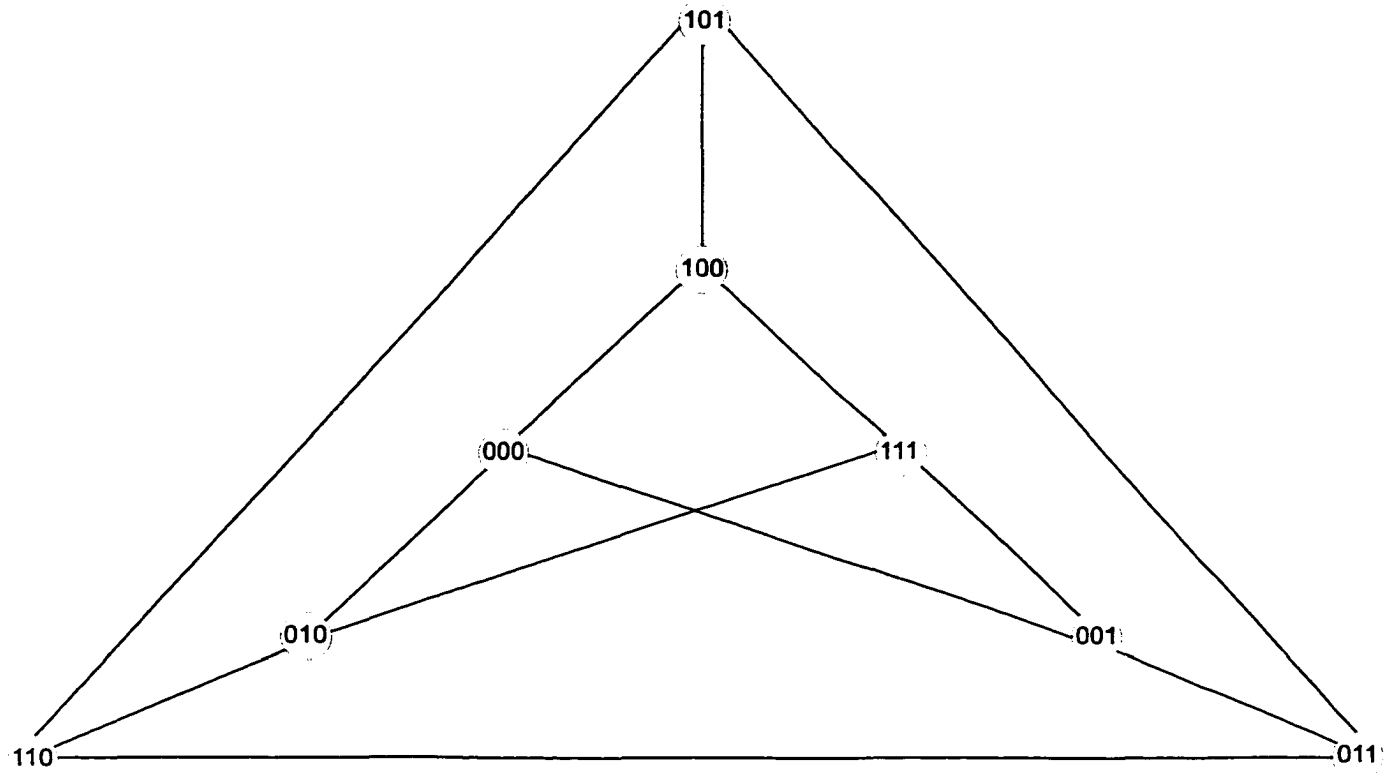


Figure 32: A 3-fast cube.

The following theorem shows how to convert an inapplicable routing symbol e_i or E_i to an applicable routing sequence of length two, i.e., a sequence of two symbols.

Theorem 6.4: *If e_i is inapplicable to node X, then it can be replaced by an equivalent routing sequence E_ie_{i-1} or $e_{i-1}E_i$ which is applicable to X.*

If E_i is inapplicable to node X, then it can be replaced by an equivalent routing sequence e_ie_{i-1} or $e_{i-1}e_i$ which is applicable to X.

Proof:

(a) First note that, if e_i were applicable to X, traversal along e_i-edge would complement bit a_i, i.e., if $X=x_{n-1}\ldots x_i\ldots x_0$, the label of the node reachable by routing symbol e_i is $Y=x_{n-1}\ldots x^c{}_i\ldots x_0$. Thus, we have to show that one of the routing sequences E_ie_{i-1} or e_iE_i is applicable to X and connects node $X=x_{n-1}\ldots x_i\ldots x_0$ to the node $Y= x_{n-1}\ldots x^c{}_i\ldots x_0$. Since $S(e_i)=S(E_ie_{i-1})=S(e_{i-1}E_i)$, these sequences are equivalent, implying that they connect node X to the node Y. Now assume that e_i is inapplicable to the node $X=x_{n-1}\ldots x_i\ldots x_0$. This implies that $x_{i+1}=1$, thus E_i is applicable to X. Traversal along E_i-edge from node X complements bits x_i and x_{i-1}.

If $x_i=0$, then e_{i-1} is applicable to X, so the string $e_{i-1}E_i$ is applicable to X and connects node X to the node Y. If $x_i=1$, then application of E_i complements bit x_i, making $x_i=0$. Now, since $x_i=0$, e_{i-1} becomes applicable to X, making E_ie_{i-1} applicable to X.

The primary reason for choosing the redundant bases $R=\{e_i, E_i : 0 \leq i \leq n-1\}$ is that, according to Theorem 6.1, any two nodes of the network can be connected by a path of length k which is, in most, cases upper bounded by n/2. In other word, the diameter of a n-fastcube is $\lceil n/2 \rceil$.

5.1 Routing in Fastcubes

This section presents two routing algorithms. The first algorithm is applicable to the n-fastcube and is a modified version of SuperCube Router. The Second routing algorithm produces an optimal shortest path in a fastcube.

Since a n-fastcube is a subgraph of an n-dimensional Supercube, the first algorithm establishes a lower bound on the diameter of a n-fastcube, which is $\lceil n/2 \rceil$.

It is interesting to note that by generating routing sequences, rather than a sequence of node labels, for connecting a pair of nodes, the routing and embedding algorithms become node independent. In other words, the routing sequences generated by the Algorithms in this chapter are applicable to all pair of nodes with the same Hamming distance.

As a result, it is necessary to run these types of algorithms, which we refer to them as node-independent, only once and used their outputs independent of the initiating node labels.

As will be discussed in the next section, the node independence property of routing sequences is especially an important property with regards to the development of embedding algorithms.

Algorithm 5: ***Fastcube Router***

Input: H(X,Y) the Hamming distance between source-destination pair of nodes;

Output: The set of routing sequences {p};

Call Algorithm 4;

Rearrange routing sequence p in descending order /the routing symbol with the least index appears on the left/

If any routing symbol is inapplicable to a source node, replace it by its equivalent routing sequence obtained from the Theorem 6.2.

Output all permutations of p.

To improve the performance of the Algorithm Fastcube, we next introduce the maximal consecutive inapplicable routing sequences (MCIRS), and show that by changing one routing symbol of a MCIRS, the whole routing sequence becomes applicable.

Definition 6.14: *Let T be a routing sequence which connects node X to node Y in a fastcube. Furthermore, assume that with respect to node X, T is composed of inapplicable routing symbols defined over the redundant set* $R = \{e_i, E_i : 0 \leq i \leq n-1\}$. *T is said to be a maximal consecutive inapplicable routing sequence (MCIRS) if and only if any pair of consecutive symbols of T is one of the following:* E_iE_{i-2}, E_ie_{i-2}, *or* e_iE_{i-1}.

For example, T= $e_5E_4E_2e_0$ is a MCIRS, and T'= $e_5e_3E_2$ is not.

The main property of a MCIRS is stated in the next theorem.

Theorem 6.5: *Let T be a MCIRS of length m. Then, there exist a sequence T' of length m+1 which is applicable to the source node and is equivalent to T.*

Proof: By construction. Let T=$t_{m-1}...t_i...t_0$ be a MCIRS defined over the redundant set R= $\{e_i , E_i : 0 \leq i \leq n-1\}$. We show that T' is obtained by replacing routing symbol t_{m-1} of T by its equivalent applicable routing sequence of length 2 (Theorem 6.4). Thus, T' and T have identical routing symbols t_{m-2} through t_0. Since the routing symbol t_{m-1} is replaced by a sequence of length 2, the length of T' is m+1.

To prove that T' is applicable to the source node S=$s_{n-1} \ldots s_i \ldots s_0$, we show that for any pair of consecutive and inapplicable routing symbols, if the left most routing symbol is executed (by replacing it by an applicable sequence), that makes the next routing symbol applicable. In general, execution of the leftmost routing symbol of any MCIRS makes all of the remaining symbols applicable. Without loss of generality, assume that the routing sequence T changes S from position s_i to s_j.

Case 1: $t_{m-1}t_{m-2}=E_ie_{i-2}$. Since by assumption, E_i and e_{i-2} are not applicable to node S, $s_{i+1}s_i$ s_{i-1} must be 011. By replacing routing symbol E_i by its applicable equivalent sequence (Theorem 6.2), s_{i-1} becomes 0 and as a result e_{i-2} becomes applicable.

Case 2: $t_{m-1}t_{m-2}=E_iE_{i-2}$. In this case, $s_{i+1}s_i\ s_{i-1}$ must be 000 or 010. Again, by replacing E_i by its applicable equivalent sequence and executing it, s_{i-1} complements and that makes E_{i-2} applicable to S.

For example, Let X=10101010 and Y=10010101. Then H(X, Y)=00111111. The minimal expansion of H(X, Y) is $\{E_1, E_3, E_5\}$. Since E_1, E_3, E_5 are not applicable to X, $T=E_5E_3E_1$ is the MCIRS. Once E_5 is replaced by its equivalent and applicable sequence e_5e_4 (Theorem 6.4), A is linked to the node Z=10011010. Since E_3 is now applicable to Z, it connects Z to the node W=10010110. Finally, execution of E_1, which is now applicable to W, connects W to the destination node Y. In this example, $T'= e_5e_4E_3E_1$.

In the next theorem, which is based on the theorem 6.1, we find the diameter D(n) of a n-fastcube and in the next section, we develop a routing algorithm which finds a shortest path of length less than or equal to D(n) between any source-destination pair of nodes.

Theorem 6.6: *The diameter of a n-fastcube, D(n), is upper bounded by* $\lceil (n+1)/2 \rceil$

Proof: The largest expansion set of H(X, Y) can be constructed as follows. Let k be the number of routing symbols in E(H(X, Y)). Furthermore, assume that non of these routing symbols are consecutive and they all are inapplicable to node X. By Theorem 6.1, each routing symbol should be replaced by a routing sequence of length 2. This makes the total length of routing sequences to be 2k. Since the routing symbols are not consecutive, they

are separated by at least 1 bit. Thus, the total number of bits between routing symbols is k-1. Taking into account that each routing sequence changes two bits of source node, one can find k as an integer satisfying equation $(2k)/2 + (k-1) \geq n$, or $k \geq (n+1)/2$.

5.2 An Optimal Router for the Fastcube.

In this section, we develop two algorithms. The first algorithm generates the fastcube topology for a given dimension n. We then present an optimal router for the n-fastcube.

The following algorithm takes as input the node label $X = x_{n-1}\ldots x_1 x_0$ and a position i within X (the ith dimension of X), and outputs the neighbor of X along that dimension.

Algorithm- NEIGHBOR:

If $x_i == 0$

Output $x_{n-1}\ldots x_i x'_{i-1}\ldots x_1 x_0$

Else

Output $x_{n-1}\ldots x_i x'_{i-1} x'_{i-2}\ldots x_1 x_0$

Note: If n=3 and the label of X is 110 → x_1=1, then the neighbor of X along the first dimension is 011.

For example, consider node X = 0101 of a 4-fastcube. Here are the four neighbors of X:

x_3=0, the neighbor along the 3rd dimension is 0001.

X_2=1, the neighbor along the 2nd dimension is 0110.

X_1=0, the neighbor along the 1st dimension is 0100.

X_0=1, the neighbor along the 0th dimension is 1101.

The neighboring algorithm defines the topology of the n-fastcube. The node set is the same as the n-cube node set, modeled by the recurrence equation $X_n = 2\ X_{n-1}$, or in other words $N = \{X:X \in B^n\}$. The edge set is defined as follows $E = \{(X, Y)|\ X \in N$, and $Y =$ Neighbor $(X, i): 0 \leq i < n\}$.

The fastcube that is spawn by the redundant basis $R= \{e_i\ , E_i :\ 0 \leq i \leq n-1\}$ has the diameter of $\lceil (n+1)/2 \rceil$, which of course is a far shorter than the diameter of the same size cube. In other words, in a 12-cube, it takes, in the worst case, 12 steps to setup a path from between a source-destination pair of nodes. In a 12-fastcube, it will take at most 7 steps to setup the path. Thus, the communication latency of a fastcube is substantially less than that of an n-cube.

The next algorithm is an optimal routing algorithm for the fastcube. This algorithm is a recursive type algorithm that takes as input a pair of source-destination labels and searches for a path from the source to the destination. The route, i.e., the labels of

intermediate nodes, is stored in a variable r. The algorithm terminates when the source label is equal to the destination label.

The algorithm starts by setting r to null and comparing the source and the destination labels. If they are equal, the algorithm returns true, and r contains the route. Otherwise, there is at least one link to get to the destination. In this case, the algorithm generates all the neighbors of the source node, using algorithm neighbor, and calls the algorithm recursively by moving the source one level closer to the destination node. Eventually, at a level less than the diameter, the source will be equal to the destination, and r will contain all the intermediate nodes. In this algorithm, d denotes the diameter of the n-fastcube.

Algorithm-OPTIMAL ROUTER:

Input:

Source node: source.

Destination node: destination.

A integer variable to hold the depth of the search initially set to 0.

Variables:

An empty string to hold all the intermediate nodes: r = empty.

A local variable Reached that will reflect the value true if the destination has bee reached, false otherwise.

Output:

The route from source to destination will be stored in r.

Return value:

A Boolean value, true or false.

Boolean Route (Source, Destination, level)

//This variable is a local variable used to see if the destination is reached.

Var Reached

Begin

//Assume that the destination was not yet found

Reached = false

//If the depth of the tree is more than the diameter, certainly the //search has gone the wrong way, so return false meaning that the correct //route is not along this branch of the tree

If (level > d)

Return false

If (source is not equal to destination)

Begin

//Generate the neighbors of the source one a time and repeat the //above process, searching for the destination.

For i = d-1 down-to 0

Begin

//If the destination has been reached (source = destination), //set reached to true and interrupt the search. This will //allow break (end) the loop, and allow the recursion to //complete its job (to fill r) with the route.

If (route (Neighbor (Source, i), Destination, level+1) = true)

Begin

Reached = true

Interrupt the search

End

End

End

If (Not Reached) return False

$R = r \cup source$

Return true

End

Algorithm-OPTIMAL ROUTER indicates that there exists a tree rooted at every node of the n-fastcube with a depth of $\lceil (n+1)/2 \rceil$.

The above algorithm is best understood by means of examples. We will illustrate how the above algorithm works in by means of the two following two examples. The algorithm searches for the destination node by taking the source label and generating a neighbor. If the generated neighbor is the destination, the search is over. Otherwise, the same process is repeated to the newly generated neighbor. Of course the depth of the search can not exceed $\lceil (n+1)/2 \rceil$. So, if the depth is exceeded, the search takes a different direction, i.e. a different neighbor is generated and a search is started in that direction. The algorithm searches for the destination in the tree spanned by the n-fastcube and whose depth is $\lceil (n+1)/2 \rceil$.

Example1:

In a 3-fastcube, a path to be establish between node X=000 and node Y=111. Following are all the recursive calls that will lead to the shortest path between X and Y. We will denote by N (X, i) the neighbor of X along the i^{th} dimension. The diameter d of the 3-fastCube is $d = \lceil (n+1)/2 \rceil = 2$.

000 111 0: src=000, dest=111, (level=0)≤d.

010 111 1: src=N (000, 2)=010, dest=111, (level=1)≤d, src≠dest.

000 111 2: src=N (010, 2)=000, dest=111, (level=2)≤d, src≠dest.

010 111 3: src= N(000, 2)=010, dest=111, (level=3)>d→Halt this search.

001 111 3: src= N(000, 1)=001, dest=111, (level=3)>d→Halt this search.

100 111 3: src= N(000, 0)=100, dest=111, (level=3)>d→Halt this search.

111 111 2: src=N (010, 1)=111, dest=111, (level=2)≤d, src=dest. Stop.

Since src=dest the route has been found, and is traced as follows:

Dest=111=N(010, 1)←010=N(000, 2)←000=src. The route therefore is: 000→010→111.

Example2:

Suppose that we want to route a message in 3-FastCube between node X=000 and node Y=111. Here are all the recursive calls that will lead to the shortest path between X and Y. We will denote by N (X, i) the neighbor of X along the i^{th} dimension. The diameter d of the 3-FastCube is $d = \lceil n/2 \rceil = 2$.

010 111 0: src=010, dest=111, (level=0)≤d, src≠dest

000 111 1: src=N (010, 2)=000, dest=111, (level=1)≤d, src≠dest.

010 111 2: src=N (000, 2)=010, dest=111, (level=2)≤d, src≠dest.

000 111 3: src=N(010, 2)=000, dest=111, (level=3)>d→Halt this search.

111 111 3: src=N(010, 1)=111, dest=111, (level=3)>d→Halt this search.

110 111 3: src=N(110, 0)=110, dest=111, (level=3)>d→Halt this search.

001 111 2: src=N (000, 1)=001, dest=111, (level=2)≤d, src≠dest.

011 111 3: src=N(001, 2)=011, dest=111, (level=3)>d→Halt this search.

000 111 3: src=N(001, 1)=000, dest=111, (level=3)>d→Halt this search.

111 111 3: src=N(001, 0)=111, dest=111, (level=3)>d→Halt this search.

100 111 2: src=N (000, 0)=100, dest=111, (level=2)≤d, src≠dest.

111 111 3: src=N(100, 2)=111, dest=111, (level=3)>d→Halt this search.

101 111 3: src=N(100, 1)=101, dest=111, (level=3)>d→Halt this search.

000 111 3: src=N(100, 0)=000, dest=111, (level=3)>d→Halt this search.

111 111 1: src=N (010, 1)=111, dest=111, (level=1)≤d, src=dest. Stop.

Since src=dest the route has been found, and is traced as follows:

Dest=111=N(010, 1)←010= src. The route therefore is: 000→010.

The run-time of the above algorihtm can be improved by observing that N(N(X, I), I) = X and thus, we can eliminate several computations.

Improved Algorithm-OPTIMAL ROUTER:

Input:

Source node: source.

Destination node: destination.

An integer variable to hold the depth of the search initially set to 0.

An integer variable to hold the dimension used to compute the neighbor of this recursive call, to avoid redundant computations, initially set to -1 since no neighbors were yet generated.

Variables:

An empty string to hold all the intermediate nodes: r = empty.

A local variable Reached that will reflect the value true if the destination has bee reached, false otherwise.

Output:

The route from source to destination will be stored in r.

Return value:

A Boolean value, true or false.

```
Boolean Route (Source, Destination, level, Dimension-Of-This-neighbor)
//This variable is a local variable used to see if the destination is reached.
Var Reached
Begin
        //Assume that the destination was not yet found
        Reached = false
//If the depth of the tree is more than the diameter, certainly the //search has gone the
wrong way, so return false meaning that the correct //route is not along this branch of the
tree
```

If (level > d)

Return false

If (source is not equal to destination)

Begin

//Generate the neighbors of the source one a time and repeat the //above process, searching for the destination.

For i = d-1 down-to 0

Begin

// if Dimension-Of-This-neighbor = i means that the neighbor was already computed, skip the remaining of the loop, and start next iteration of the loop.

if(Dimension-Of-This-neighbor = i) continue

//If the destination has been reached (source = destination),

//set reached to true and interrupt the search. This will

//allow break (end) the loop, and allow the recursion to //complete its job (to fill r) with the route.

If (route (Neighbor (Source, i), Destination, level+1, i) = true)

Begin

Reached = true

Interrupt the search

End

End

End

If (Not Reached) return False

$R = r \cup source$

Return true

End

Example1:

To route 000 to 111

000 111 0 -1: src=000, dest=111, (level=0)≤d.

010 111 1 2: src=N (000, 2)=010, dest=111, (level=1)≤d, src≠dest.

111 111 2 1: src=N (010, 1)=111, dest=111, (level=2)≤d, src=dest. Stop.

Since src=dest the route has been found, and is traced as follows:

Dest=111=N(010, 1)←010=N(000, 2)←000=src. The route therefore is: 000→010→111.

Example2:

To route 010 to 111

010 111 0 -1: src=010, dest=111, (level=0)≤d, src≠dest

000 111 1 2: src=N (010, 2)=000, dest=111, (level=1)≤d, src≠dest.

001 111 2: src=N (000, 1)=001, dest=111, (level=2)≤d, src≠dest.

011 111 3: src=N(001, 2)=011, dest=111, (level=3)>d → Halt this search.

111 111 3: src=N(001, 0)=111, dest=111, (level=3)>d → Halt this search.

100 111 2: src=N (000, 0)=100, dest=111, (level=2)≤d, src≠dest.

111 111 3: src=N(100, 2)=111, dest=111, (level=3)>d → Halt this search.

101 111 3: src=N(100, 1)=101, dest=111, (level=3)>d → Halt this search.

111 111 1: src=N (010, 1)=111, dest=111, (level=1)≤d, src=dest. Stop.

Since src=dest the route has been found, and is traced as follows:

Dest = 111 = N(010, 1) ← 010 = src. The route therefore is: 000 → 010.

Chapter 7

Fault Tolerance

In this section we investigate some additional properties of LRNs which are related to their fault tolerance. The system detects problems usually by means of some mechanism that runs periodically making sure that the system is functional. The mechanism usually sends probe messages to processing elements and waits for replies within well-defined time outs. If no response is received from a processing element, that processing element is considered faulty and the system needs then to be reconfigured to work properly with the rest of the processing elements or a subset of them. A response might not be received for these main reasons:

The designated processing element never received the probe.

The designated processing element received the probe but did not process it.

The designated processing element received the probe, processed it, and replied back, but the reply never reached the destination.

Reasons (a) and (c) are of the same nature and are due to faulty links. Reason (b) is due to a faulty processing elements.

Failures can be divided into two groups. Failures can be due to faulty processing elements or faulty links. A faulty processing element is a processing that is no longer functional. This processing element can no longer receive, send nor forward messages. It also can not execute any Jobs. Such a processing element must be removed or bypassed

until it is functional again. If there is a message travelling from node X to node Y, and node Z, a non-functional processing element, is along the path from X to Y, then the message will never get to Y. The integrity of the system will be disrupted and the system will no longer do its tasks properly. Sometimes the processing element is not down but might be used for some specific purpose and can not be used as intermediate vertices of paths.

Failures can also be due to faulty links or lines that connect nodes. Here one or more edges are disconnected or do not interconnect their end point processing elements correctly. The chances are that processing elements go down much more frequently than lines. But it is worth the attention to investigate how to cope with these kinds of failures. Processing elements connected with faulty lines will remain operational as long as there is at least one undamaged line left. If all the lines (wires) that connect this processing element to the rest of the network are faulty, than the processing element is no longer accessible by its peers in the network and can be considered as being faulty.

A fault tolerant network is one that can still function fully or partially even in case of failures. Ideally, if a node is down, it is not to be involved in any processing or messaging. That is no jobs are assigned to that processing element, and it does not serve as an intermediate node along the path between two functional nodes. If a link is down, that link is no longer used. That is, paths involving that link are no longer valid. Some alternative equivalent path needs to be determined and used for routing. A node failure also requires finding alternative paths, since the path involving the faulty node is no longer valid.

Sometimes, a node failure might imply the failure of a sub-system of the network, or of the entire network, if the processing element in question is a central processing element although each processing element remains individually functional.

Failure detection is beyond the scope of this thesis, we will investigate in this chapter how to take a faulty LRN and reconfigure it so that the resulting one is functional. We will use some results from chapter 5 (Embedding and partitioning) to demonstrate that in case of failure of a LRN $G_n(A)$ due to faulty processing elements, a sub-network $G_n(A')$ could remain functional such that $G_n(A') \subseteq G_n(A)$ so that those faulty processing elements are not in the vertex set of $V_n(A')$.

1 Node failures

Node failures are the most common reason that results in the failure of an interconnection network. In the case of LRNs, since the any LRN can be decomposed in terms of lower dimension networks, we can reconfigure the network in such a way that the faulty node no longer participate in the operation of the entire network. This can be accomplished in two distinct ways:

Finding a LRN that is a sub-network of the original network involving most of the nodes except the faulty node, or a small subset of nodes (including the faulty node). We call this method *subnetting method.*

Decomposing the LRN and bypassing the cluster to which the faulty node belongs. We will call this method *partitioning method.*

1.1 Subnetting method

A LRN could no longer be functional because of a single node failure or a multiple node failure. Applying over and over the subnetting method to a single node solves the multiple node failure as will describe later in this section.

Let $A = f_1(n).f_2(n)...f_k(n)$ be the generator of a LRN. The purpose of the subnetting method is to find A' the generator of a new LRN so that the new labels generated from A' do not include the faulty node (or nodes) and such that A' < A. The choice of A' has to be such that $X_n(A')$ is maximal. This will be accomplished by first decomposing the label of the faulty node according to definition 3.7 in a form such as $A_x\ A_{x'}$.... Then we have to find k = max (x, x'...) and

$$A' = \begin{cases} a_1...a_{k-1}(a_k - 1), if\ (a_k - 1) > 0 \\ a_1...a_{k-1} \qquad \text{Otherwise} \end{cases}$$

This choice of A' guaranties both A' < A so that $G_n(A') \subseteq G_n(A)$, the faulty node, v, will no longer belong to $V_n(A')$ and $X_n(A')$ is maximal. The subnetting method is best understood using the following example:

Let A = 2.3.5, → A_1 = {0, 1}, A_2 = {2.0, 2.1, 2.2}, A_3 = {2.3.0, 2.3.1, 2.3.2, 2.3.3, 2.3.4}.

$V_4(A)$ = {0.0.0, 0.0.1, 0.1.0, 0.1.1, 0.2.0, 0.2.1, 0.2.2, 1.0.0, 1.0.1, 1.1.0, 1.1.1, 1.2.0, 1.2.1, 1.2.2, 2.0.0, 2.0.1, 2.1.0, 2.1.1, 2.2.0, 2.2.1, 2.3.0, 2.3.1, 2.3.2, 2.3.3, 2.3.4}

Suppose that v = 2.3.2 be comes faulty, the according to the subnetting method we first have to decompose v according to definition 3.7: DEC(v) = A_3. So the best choice

according to the subnetting methods would be A' = 2.2. Other generators B such that B < A' will also allow the system to be functional again but with lesser nodes than A'.

1.2 Partitioning method

The partitioning method is a simpler method to apply since the results of chapter 5 can be used to partition the LRN and used the one to which the faulty node or nodes do not belong. But its downside is that the size of the new LRN could be considerably smaller than the original one, especially if the faulty nodes belong to different partitions.

The partitioning method works as follows. Let A be the generator of a LRN. If the LRN becomes non-functional because of a node failure, identify the label of the node. Decompose (partition) the LRN into distinct clusters using the results of chapter 5. Then identify the cluster to which the label of the faulty node belongs. This cluster can no longer be used as part of the system. The cluster can not be used to run jobs and should not be involved in any routing. The remaining functional clusters can function independently as subsystems. Interprocessor communication between these clusters can still be possible if there exists at least one link between the independent clusters, which will provide alternative paths.

The above two methods will shrink the size of the system by more than just the faulty nodes. If this is acceptable for certain systems/applications, the system needs to be reconfigured so that all the remaining functional nodes are used. This can only be accomplished by conserving the original topology and not involving the faulty nodes with any processing. Faulty nodes can not execute jobs, nor can they participate in routing. If one of the faulty nodes serves as a gateway to a group of functional nodes, that group is

detached from the remaining of the system. The latter is somewhat similar to the partitioning method

2 Link failures

Link failures can also be the cause of a system's malfunction. For example, if a system is using a static route to provide interprocessor communication and the physical link that connects the two processors fails, then that route can no longer be taken. This is usually not a fatal error in a system that consists of parallel/alternative routes.

In case of unavailability of a path, an alternative path must be considered. The alternative path should be minimal, but that is not possible all the time. The alternative path and the original path should be could be disjoint (all nodes in the alternative path should be different except for the source and the destination), but not necessarily. If the case that the paths are disjoint, they are also called parallel paths.

To illustrate the two above concepts, parallel and alternative paths, consider a 3-cube generated by A = 2. $V_n(A)$ = {000, 001, 010, 011, 100, 101, 110, 111}, and $E_n(A)$ = {(000, 100), (000, 010), (000, 001), (001, 101), (001, 011), (010, 110), (010, 011), (011, 111), (100, 110), (100, 101), (101, 111), (110, 111)}. Figure 33 shows the graph of the 3-cube.

There are 2 parallel paths between 0.0.0 and 1.1.1:

p_1 = 0.0.0 → 0.0.1 → 0.1.1 → 1.1.1

p_2 = 0.0.0 → 1.0.0 → 1.1.0 → 1.1.1

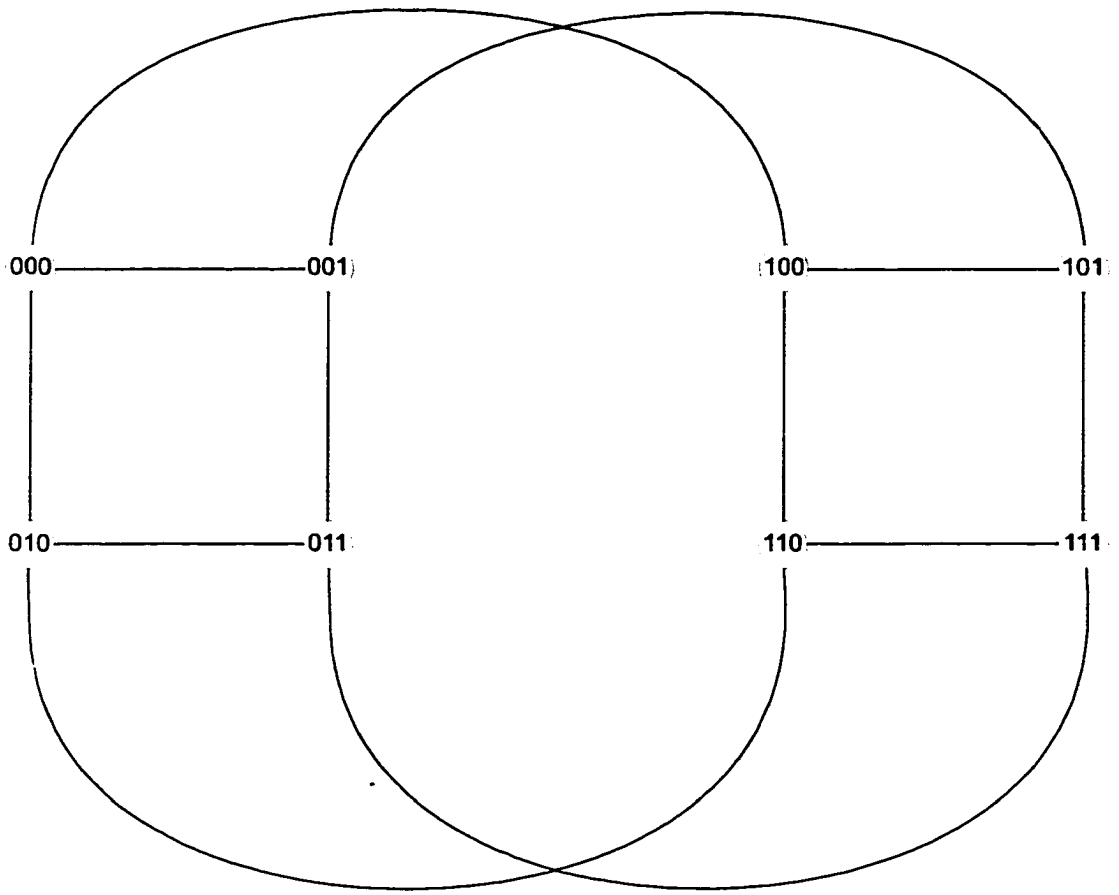


Figure 33: 3-Cube generated by A = 2.

p_1 and p_2 are parallel paths, in that all nodes in p_1 are be different from the nodes in p_2 except for the source and the destination.

Suppose that 0.0.0 → 0.0.1 was unavailable in p_1, then an alternative path must be taken. We could for example follow the following path: p_3 = 0.0.0 → 0.1.0 → 0.1.1 → 1.1.1. p_3 is an alternative path to p_1 but not parallel to p_1. On the other p_3 is parallel to p_2.

p_3 bypasses the edge (0.0.0, 0.0.1) since it is faulty. It uses instead the edge (0.0.0, 0.1.0) that will eventually route it to .0.1.1.

Definition 7.1: We say that a LRN, generated by A, is x fault tolerant if the graph $G_n(A)$ of a LRN is still connected even if x-1 links fail.

Lemma 7.1: Let r be the minimum node degree in the graph $G_n(A)$ of an LRN generated by A. HLRNs are r fault tolerant.

Proof:

The worst scenario possible is that all the r-1 links that failed are applied to the same node, in which case, it will still be attached to the rest of the nodes via the last remaining functional link. Thus the resulting graph $G'_n(A)$ is a connected graph.

Chapter 8

Conclusion

In this thesis, we present a modeling technique to study the class of linear recursive networks, LRNs. The proposed modeling technique can also be applied to analyze most interconnection networks, and to design new interconnection topologies.

This chapter will summarize the results from this thesis. It demonstrates that the proposed modeling technique is a valuable tool to the design and analyses of interconnection networks as well as communication networks. We will also present a list of open problems that we feel need to be further researched to better understand LRNs. A complete list of results, namely definitions, theorems, corollaries, lemmas, algorithms ... can be found in the appendix.

1 Summary of the thesis and results

In this thesis, we presented an extensive study of LRNs and their variances. All interconnection networks whose size satisfies a linear recurrence equation are members of the LRN class.

Chapter 1 introduces interconnection networks. Common interconnection topologies are presented and some of their drawbacks are pointed out.

Chapter 2 describes the motivation behind this research. , It presents the current literature in the subject of LRNs. We then pointed out improvements/changes that can enhance the class of LRNs.

In Chapter 3, we modified the definition of LRNs presented in [23] to cover a wider range of interconnection topologies. We presented the improved definition of the modeling technique used to model interconnection networks: LRM. One of the most important properties of the modeling technique is the separation between the node set and edge set of a network. Some of the properties of LRNs controlled by the LRM are the size of the network and its scalability. The choice of the interconnection method is a separate process and is heavily influenced by other parameters such as cost and performance. Many static properties of LRNs, such as node labels and their inherent uniqueness due to LRM are studied in this chapter. An algorithm to decompose node labels is developed and formally presented. This algorithm is used in a later chapter to study fault tolerance of LRNs. Complexity analysis for the decomposition algorithm was presented. An upper bound of the size of all LRNs was given. We then presented a classification of all LRNs, which demonstrates that the LRN class is a large class of interconnection networks and that many of the common topologies are LRNs. A very important subclass of the LRN class was introduced: the *HLRN* class. The HLRN class is a class of LRNs whose members' nodes are interconnected by a hamming distance. We proved that the graph of any HLRN is a connected graph, and we proposed that the diameter of any HLRN is bounded above by n. We then presented in chapter 3 several approaches to selecting the interconnection method. One way of interconnecting the nodes lead to show that the Hypercube is a HLRN using Hamming distance of one as its

interconnection method. Another method would be used in chapter 6 to develop the n-fastcube.

Chapter 4 is devoted to the routing in LRNs. We first reviewed a well-known static algorithm that is a solution to the single-source shortest-paths problem. We used it later in the chapter to develop algorithms that are more efficient. We identified three types of routing algorithms. The first types of routing algorithms are the *static routing algorithms*. Static routing algorithms are run off-line. Tables are constructed and used later to route messages between nodes. This is very similar to routing found in gateways used in LANs. A drawback of most static algorithms is the need of storage and tables lookups. Also they are not very suitable for dynamic reconfiguration. The second types of routing algorithms presented in chapter four are the *semi-dynamic algorithms*. Semi-Dynamic algorithms are those algorithms that use static information about the network only once to figure out a way to route messages such as computing routing functions. Once the system is functional, these algorithms do not require any substantial storage, nor do they require table lookups. These algorithms no longer need the static information, and behave as though they were dynamic algorithms. Three semi-dynamic algorithms were developed: The *Switched Algorithm:* an application to switching theory, The *Tree algorithm* and The *Transition algorithm*. A complexity analysis was presented for each of the above mentioned algorithms. The third types of routing algorithms are the *dynamic algorithms*. Dynamic algorithms were not developed in chapter 4 since they are topology dependent.

We demonstrated in chapter 4 that routing algorithms correspond to a certain layer in the hierarchy of LRN. Therefore, routing algorithms are applicable to all those interconnection networks that belong to that branch of the hierarchy. This is an important

property of LRNs since algorithms do not have to be developed for individual topologies. Algorithms can be developed for a wide range of topologies by only using common properties to all the members of a certain layer. For example, The switching algorithm is defined at the topmost layer of the LRN hierarchy and therefore can be used to route messages in any LRN. On the other hand, the Tree algorithm is defined on a lower layer of the hierarchy. It can only be used to route messages in HLRNs. The need of developing routing algorithms occurs when the performance of existing routing algorithms is no longer acceptable.

Chapter 5 discusses embedding *properties* of LRNs. We first present two partial ordering relations. One is used to obtain more information about a specific LRN. This information pertained to the number of specific Lower dimension network that constitutes the current dimension. In other words given $G_n(A)$ how many copies of $G_x(A)$ exist in $G_n(A)$ where x is less than n. Let that number be m. Therefore we can embed m copies of $G_x(A)$ in $G_n(A)$. The second partial ordering was the main tool that we used to show how LRNs could embed other topologies and how we could partition them. We were able to define necessary and sufficient conditions for embedding LRNs. We also showed how an n-cube could be embedded into an n-LRN. Embedding of rings was also presented. Partitioning of LRNs, which is essential in multiprocessing environment, was discussed. It was shown that the partitioning process is controlled by the generator of the LRN, A. A very important property of graphs was studied, namely the Hamiltonian property. We presented sufficient conditions under which a HLRN is Hamiltonian.

Chapter 6 presents a new cube-like interconnection topology, *n-fastcube*. The topological and routing properties of fast-cubes are investigated in this chapter. The diameter of the

fastcube if almost half of the diameter of the same size cube. In the same context, a methodology to reduce the diameter of interconnection networks is introduced. The n-fastcube is an LRN that has the same cost as the n-cube but a lower diameter. The latter is a perfect example that supports the LRM of how the LRM can be useful in developing new topologies, possibly more efficient than existing topologies.

Chapter 7 discusses fault- tolerance of LRNs. Ways to handle node failures are presented. Two methodologies are developed to deal with node failures: The *subnetting method* and the *partitioning method*. It is demonstrated that LRNs are inherently fault tolerant. Link failures were also briefly discussed.

All the above results hint that the LRM is an efficient tool in studying interconnection networks. LRNs have very attractive properties and deserve additional research. There are many problems that are not covered in this thesis and that should be explored. Other problems discussed in this thesis should be further studied to get a better understanding of LRNs. The next section presents a list of open problems.

2 Open Problems

The first problem conjectured in chapter 3 is that the diameter of any HLRN is bounded above by n. This intuitive result has not been formally proven. A formal proof would generalize the HLRN class and make it subject of much attention because of its small diameter. The lower bound on the node degree of HRLN has not been determined. We stated that if r is the minimum node degree in a HLRN, then any HLRN could survive r-1

link failures. Figuring out the exact value of r is very important in studying the fault tolerance of HRLNs.

Though many routing strategies were developed in this thesis, optimal dynamic routing strategies defined at high levels of the hierarchy are still open problems. This will allow for optimal routing for all members of LRNs. It seems that it is possible to develop optimal dynamic routing strategies for the HLRN class. The decomposition method presented in chapter 3 seems to be a suitable tool in developing above algorithms.

We gave most attention to the HLRN class. Other interconnection methods should be investigated to possibly improve cost/performance of interconnection networks. Also other interconnection methods should be investigated to possibly develop new topologies for the same class of LRNs, as chapter 6 demonstrated.

Embedding was defined in terms of generators. If $B \leq A$, then $G_n(B)$ can be embedded in $G_n(A)$. This result presents a sufficient condition but not a necessary one. Further, there might be other LRNs that can be embedded but do not respect the above condition. More work needs to be done in this area to determine more flexible conditions under which networks can be embedded.

In the same area, the Hamiltonian property was explored but not totally related to the generator of the LRN.

More efficient routing algorithms need to be developed for the n-fastcube. The n-fastcube is a very attractive alternative to the n-cube, but it lacks efficient and easy routing algorithms. It also lacks embedding algorithms. Fault tolerance need to be investigated for the n-fastcube.

Link failures, alternative and parallel paths were not heavily studied in this thesis. Fault tolerance of interconnection networks is a very important issue and deserves much attention.

References

[1] Abraham S. and Padmanabhan K.: *An analysis of the twisted cube topology*, *1989 Int'l Conference On Parallel Processing*, vol. 1, pp. 116-120 Pennsylvania State Press.

[2] Akers S.B. and Krishnamurthy B.: *A group-theoretic model for symmetric interconnection networks*, IEEE Trans. On Comp. 1989, vol.38, No 4, 555-566.

[3] Akers S.B. and Krishnamurthy B.: *The Fault Tolerance of Star Graphs*, Proc. 2nd Int. Conference Supercomputing, vol. III, pp. 270-276, 1987.

[4] Bavel Z.: *Introduction to the Theory of Automata*, Prentice-Hall, New Jersey, 1983.

[5] Biggs N.L.: *Discrete Mathematics*, Royal Holloway College, University of London, Clarendon Press, Oxford, 1985.

[6] Cormen T. H., Leiserson C.E. and Rivest R.L.: *Introduction to algorithms*, The MIT press, Cambridge, Massachusetts, 1989.

[7] Cull P. and Larson S.: *The Mobius Cube: An interconnection network for parallel processing*, OSU Tec. Rpt. 91-20-2, Department of Computer Science, Oregon State University, 1991.

[8] Cull P. and Larson S.: *The Mobius Cubes*, IEEE Trans. On Computers, vol. 44, no. 5, pp. 647 659, May, 1995.

[9] Derr and Lecasseur K.: *Applied Discrete Structures for Computer Science,* Science research Associates Inc., Chicago, 1985.

[10] Efe K.: *A variation on the Hypercube with lower diameter*, IEEE Trans. Computers, vol. 40, no. 11, pp. 1312-1316, Nov. 1991.

[11] Estafahanian A.H., Ni I.M., and Sagan B.: *The Twisted n-cube with application to multiprocessing*, IEEE Trans. Computers, vol. 40, no. 1, pp. 88-93, Jan. 1001.

[12] Feng T.Y.: *A survey of Interconnection Networks*, IEEE Computer, pp.12-27, Dec. 1981.

[13] Flynn M.J.: *Very High-Speed Computing Systems,* Proc. IEEE vol.14, pp.12-27, Dec. 19981.

[14] Gallian J.A., *Contemporary abstract Algebra*, second edition, Lexington books, D.C. Health and company, Lexington, Massachusetts, 1990.

[15] Ghozati S.A. and Smires T.: *The Fastcube: A Variation on Hypercube Topology with Lower Diameter*, Journal of Computers and Electrical Engineering Journal, Submitted Apr. 1999.

[16] Ghozati S.A. and Wasserman H.: *On Routing in Fibonacci Graphs*, Journal of Networks, Under review, Submitted 1998.

[17] Ghozati S.A. and Wasserman Howard: *The k-ary n-cube Network: Modeling, Topological Properties and Routing*, Journal of Computers and Electrical Engineering Journal, Accepted Oct. 1998.

[18] Ghozati S.A.: *A New Methodology for the Design and the Analysis of a Class of Communication Networks*, International Journal of Computers and Electrical Engineering Journal, vol.21, issue no. 5, pp. 341-349, 1995.

[19] Ghozati S.A.: *A Unified Approach to Interconnection Networks: Models, Algorithms and Analysis*, Journal of Computers and Electrical Engineering Journal, vol. 23, pp. 135-150, 1999.

[20] Ghozati S.A.: *High Speed Communication Networks*, IASTED, Reliability, Quality Control and Risk Assessment, ACTA Press, p.163-167, Oct. 1993.

[21] Hayes J.P.: *Computer Architecture and organization*, McGraw-Hill, New York, 1988.

[22] Hilbert P.A.J., Koopman R.J.M., and Van de Snepscheut J.L.A.: *The Twisted Cube, Parallel Architecture and Languages Europe*, vol. 1, Parallel Architecture, pp. 152-158, J. deBakker, A Numan, and P. Trelearen (Eds), Berlin Springer-Verlag, 1987.

[23] Hsu W., Chung M.J. and Das A.: *Linear Recursive Networks and Their Applications in Distributed Systems*, vol. 8, no. 7, pp. 673-680, July 1997.

[24] Leighton F.T.: *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufman Publishers, San Francisco, California, 1992.

[25] Siegel H.J.: *Interconnection Networks for Large-Scale Parallel Processing*, Lexington books, D.C. Health and company, Lexington, Massachusetts, 1985.

[26] Singhvi S.K. and Ghose K.: *The M-Cube*, Technical Report No. CS-TR-91-10, Dept. of Computer Science, State University of New York, Binghamton, 1991.

[27] Skvarcuis R. and Robinson W.B.: *Discrete Mathematics with Computer Science Applications*, The Benjammin/Cummings Publishing Company Inc., New York, 1986.

[28] Stallings W.: *Data and Computer Communications*, second edition, Macmillan Publishing Company, New York, 1988.

[29] Tanenbaum A.S.: *Distributed Operating Systems*, Prentice Hall, Upper Saddle River, New Jersey, 1995.