Multidimensional Linear Congruential Graphs: A New Model for Large-Scale Interconnection Networks

Ching-Chun Koung

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

June 1993

ISBN 0-315-87270-5

Abstract

Multidimensional Linear Congruential Graphs: A New Model for Large-Scale Interconnection Networks

Ching-Chun Koung

A multicomputer offers high degrees of concurrency in computation and therefore provides an effective solution to many applications that demand complex and fast computing power. The interconnection network in a fine-grain multicomputer connects thousands small computers and its design is an important part of such a multicomputer system.

In the design of an interconnection network, graphs are usually used to model the network, and the question whether a graph satisfies the requirements of an interconnection network can be decided by examining the graph's topology.

A Linear Congruential Graph of degree $2d$ is a graph on the vertex set $\{0, 1, \ldots, n-1\}$ whose edge set is generated by a set of generators $\{f_1, f_2, \ldots, f_d\}$ where each generator is a linear function.

We generalize the linear congruential graph into the multidimensional case in which the vertex set consists of vectors of integers and generators are linear functions on the vectors. We show that

1. there are 2-dimensional linear congruential graphs of degree Δ and diameter D which contain more vertices than linear congruential graphs for many values of Δ and D.

2. the properties of two-dimensional linear congruential graphs meet the requirements of a large-scale network.

3. there are sufficient conditions on the two-dimensional generators to generate Hamiltonian cycles and edge-disjoint cycles.

Comparisons are given between these graphs and some well known families of graphs for their network properties.

Acknowledgement

First and foremost, I would like to express the deepest gratitude to my thesis supervisor, Dr. J. Opatrny, for the support and guidance he has given me during the course of this thesis. I am very grateful for the consideration and encouragement he has offered during my studies. I also appreciate his patience and understanding when things were frustrating.

I would further like to thank Mr. Raymond Lin, who is working on the routing algorithm of this network model, for his help and friendship. I am impressed by his intelligence and discussions which are always fruitful.

Finally, thanks to my parents for their confidence and support from the beginning.

Dedicated to my parents

Contents

List of Figures

List of Tables

List of Lemmas

Chapter 1

Introduction

1.1 Problem Domain

Communication networks have been used in many applications of our everyday life. In the area of computer science, networks have been used to connect a number of computers (processors) to fulfill tasks. A variety of popular and promising systems have been built: LAN, MAN, WAN, internet, distributed systems, and multicomputer systems. Each of the above systems contains more than one computer (processor), and a network is used to connect these computers (processors). Moreover, the networks impose many performance restrictions in these systems.

In the design of a computer network, graphs are usually used to model the computer network in which computers (processors) are represented by vertices and communication links are the edges of a graph. The necessary requirements of a good network are therefore corresponding to the desirable properties of a graph. Whether a graph satisfies the requirements of a good computer network can also be decided by considering the topology of the graph. We will use *network models* to address those graphs that are used to model computer networks.

In this thesis, we review the issues of network design and performance. Since a multicomputer, which is designed for parallel processing, takes more strict requirements on the topology of its interconnection network, we will give a review of the techniques and systems that are used for parallel processing, in particular, multicomputer systems. The desirable graph-theoretic properties of a multicomputer network model are then derived. Four important network models are also reviewed and their graph-theoretic properties are given. We then propose a new network model, multidimensional linear congruential graphs.

We will show that our multidimensional linear congruential graphs have many good properties to be an alternative to large-scale interconnection networks. Properties which are specific to this model are discussed. The four network models reviewed before are compared with our model. It is shown that the network properties of our model are better than those of other models in many cases.

1.2 Motivation

A multicomputer offers high degrees of concurrency and fault-tolerance in computation, and has open-ended extensibility. A fine-grain multicomputer contains thousands to hundreds of thousands of small computers (nodes) and provides an efficient and cost-effective solution to many applications that demand complex and fast computing power. A task is completed in a multicomputer by partitioning the task into a number of subtasks, which are executed concurrently in distinct nodes. Interaction and cooperation among subtasks is done by sending messages to other nodes through an interconnection network, which connects all the nodes. As the number of nodes in a multicomputer increases, a larger latency of message traversal time in the network will be exhibited and it will degrade the system performance. The latency is proportional to the number of nodes traversed between a sender and a receiver. Furthermore, the number of communication links used in a node to communicate with other nodes is bounded.

Using the graph representation of a network that we mentioned earlier, it is necessary to construct graphs that have a very large number of vertices in order to model the network of a fine-grain multicomputer. Since the latency is proportional to a graph's *diameter*, which is the maximum distance among all pairs of vertices, the diameter of the graphs should be as small as possible. The number of communication links corresponds to a vertex' *degree*, which is the number of edges incident with the vertex. A graph's *degree* is the largest degree among its vertices and its *size* is the number of vertices in the graph. Thus it is important to construct graphs of large size, small diameter, and of bounded degree.

The tables in [30, 17, 8] show the progress on constructing large graphs of small degree and diameter made in the last ten years. The most recent results are in [15]. As can be seen in these tables, the construction methods used for these graphs are not uniform. A specific construction method is required for a given degree and diameter.

Investigation of random graphs showed that random graphs have low diameters in almost all cases and the diameters are very close to the optimum [10]. However, a random graph has the disadvantage that it is difficult to construct low-diameter random graphs by any explicit method [12]. It is also difficult to verify their properties because their structure is highly irregular. Therefore, to use a random graph with n vertices, it would take n^2 entries of memory to store it and the same size of memory would be required for routing in the graph [27]. On the other hand, the graphs generated by an explicit method can be analyzed systematically and use less memory, which is especially important when designing large-scale networks.

The best general constructions of large graphs of a given degree and diameter are de Bruijn graphs and their variations, such as Kautz graphs [9], and Imase-Itoh graphs [26]. The DCC linear congruential graphs in [31] generalize the construction of de Bruijn graphs, and contain more vertices than the de Bruijn graphs for the same degree and diameter. Inspired by the idea of DCC linear congruential graphs, we propose a new construction method. It generalizes the construction of DCC linear congruential graphs and hence, the de Bruijn graphs.

The aim of this thesis is therefore to provide a uniform method for constructing large-scale graphs of any degree and size, the diameters of these graphs are made as small as possible for a chosen range of degrees and sizes, and their graph-theoretic properties are suitable for computer networks.

1.3 Thesis Outline

In **Chapter 2**, we give a short review of the evolution of computer systems and the techniques for parallel processing. In particular, we review multicomputer systems, discuss the desirable properties of their interconnection networks, and relate these properties to graph theory. Network performance and its design issues are also discussed. Since our discussion will be based on graph theory, one section is devoted to the graph-theoretic terminology and properties that will be used for all network models in the thesis.

In **Chapter 3**, four important network models are reviewed and are given in separate sections. They are Hypercubes, de Bruijn graphs, Cayley graphs, and DCC linear congruential graphs. We give the definitions of these models, examine their sizes and diameters, and analyze their graph-theoretic properties. A summary is given in each section.

In **Chapter 4**, we introduce our multidimensional linear congruential graphs, which are the main subject of this thesis. An overview explains the reason why we expect this model to produce good results and then we give a formal definition of these graphs.

We will look in detail at the construction of two-dimensional linear congruential graphs and the cycle structures created in these graphs. We then examine their sizes and diameters, and analyze their graph-theoretic properties, and finally discuss those construction issues that are specific to this network model. All results in this chapter are original.

In **Chapter 5**, we compare our multidimensional linear congruential graphs with other networks based on the results from Chapter 3 and Chapter 4. It includes the issues of their construction methods, sizes, diameters, and network properties.

In **Chapter 6**, we conclude by summarizing the important results of this thesis and proposing some open problems in this research area.

Chapter 2

Computer Networks

2.1 System Evolution

Computer systems have greatly evolved over the past 50 years. From the invention of the first computer to the current experimental systems, system performance has increased 10 times every five years [23]. New hardware technology, of course, has contributed to this rapid improvement but it is not the only factor. Both architecture and system software design play very important roles in this evolution. Efforts are still being made to improve system performance in two directions. The first direction is to produce faster and more powerful hardware devices in engineering fields. The second one is to increase the degrees of parallel processing in an execution environment by proper architecture and system software design.

A formal definition of parallel processing can be found in [25]. Parallel processing provides an efficient and cost-effective means to improve system performance under the same hardware capability by exploiting events that can be executed concurrently. These events can be found at 4 different levels in an execution environment,

1. Among different programs
2. Within a program
3. Among different instructions

4. Within an instruction

After recognizing the possible concurrency that exists at these levels, different architecture and software techniques were developed to achieve parallel processing.

From the system software point of view, techniques were applied to the following operating system concepts to improve performance:

- Batch processing: Avoid CPU idle time by preloading programs into disks.
- Multiprogramming: Avoid CPU idle time by mixing the execution of CPU-bound and I/O-bound programs.
- Time-sharing: Avoid the monopolization of CPU by assigning a fixed portion of CPU time to each user.
- Multitasking: Allow a user to run more than one program at a time by background processing. A user can continue to enter commands while some programs are being executed in the background.
- Multithreading: Allow a process to have more than one thread of control. Threads in a process share the same process space and are executed concurrently.

Many operating systems. e.g., VMS, Unix, Mach [34], support more than one of the ideas described above.

From the architectural point of view, the techniques that were used to achieve parallel processing are the following:

- Pipeline processing: Instructions or arithmetic operations are executed in parallel by overlapping their execution in a processor.
- Array processing: The data of a vector operation are executed in parallel by the existence of an array of arithmetic logic units that perform the operation under the control of a central processor.

- Multiprocessing: A system that contains multiple processors. Processors can execute instructions independently and in parallel. Communication among processors can be message-passing or memory-sharing. This system includes multiprocessors and multicomputers, which will both be described later.

- Distributed processing: A system that contains multiple computers. Each com puter is an autonomous unit. It is similar to multiprocessing in that each processor executes instructions independently and in parallel. However, computers are physically distributed, communication is done only by message-passing. It is different from computer networks in that software hides the details of underlying machines from users.

Computers that can exploit parallelism in execution by their architectural designs are classified as *parallel computers.* Pipeline processing was developed on a uniprocessor system and used to improve the utilization of a single processor. The degrees of parallelism are low since jobs are still processed sequentially by the sole processor. Another class of architecture designs that use multiple arithmetic logic units or multiple processors can exploit high degrees of parallelism. It is this class of systems that we will discuss in the next section and the need of such efficient parallel computers is important because:

- They have better performance than a traditional sequential uniprocessor system in general applications.

- Data structures of certain kinds of applications, for example, image processing, simulation of VLSI, and semantic networks, fit into the structures of parallel computers much better than those of sequential systems and thus can be solved efficiently only by parallel computers.

2.2 Parallel Computers

According to Flynn's classification [25], computer systems can be classified to 4 architectural designs based on the multiplicity of instructions and data that a system can process. We give a list below and indicate the parallel processing techniques that they can use.

- Single-instruction, single-data machine (SISD): pipeline
- Single-instruction, multiple-data machine (SIMD): pipeline, array
- Multiple-instruction, single-data machine (MISD): no real application
- Multiple-instruction, multiple-data machine (MIMD): pipeline, multiprocessor, distributed processing

Based on the above categories, parallel computers consist of pipeline SISD, array SIMD, and MIMD machines.

A pipeline SISD machine can overlap the execution time among different instructions or arithmetic operations. For example, the execution of an instruction can be divided into 4 stages, instruction fetch, instruction decoding, operand fetch, and execution. Therefore, four instructions can be executed at different stages in a pipeline fashion.

An array SIMD machine contains an array of processing elements (PE) and a central control unit (CU). For vector processing, the CU distributes all data to PEs. All the PEs can execute the operation on different data in parallel under CU's control. For scalar processing, the CU can execute it directly in a pipeline fashion.

A MIMD machine contains multiple processors. It can process different instructions on different sets of data in parallel. It can also have the same functions as SISD and SIMD machines because the processors in a MIMD machine can be pipelined and an array of processors is equivalent to an array of PEs in computation. MIMD machines thus can exploit high degrees of parallelism and are expected to have better performance. Moreover, MIMD machines not only can provide high throughput but also can provide more reliable and more extensible environments than any other computer systems.

MIMD machines can be classified as loosely coupled multicomputers and tightly coupled multiprocessors. A multicomputer system is composed of a set of nodes where each one is a small computer with its local memory. There is an interconnection network that connects all nodes, as shown in Figure 2.1. Communication among these nodes is via message-passing over the network. A multiprocessor system is composed of a set of processors and a global memory module. Communication among the processors is done by memory-sharing. There is an interrupt signal interconnection network connecting all processors and a switch network connecting all processors to all memory modules, as shown in Figure 2.2.

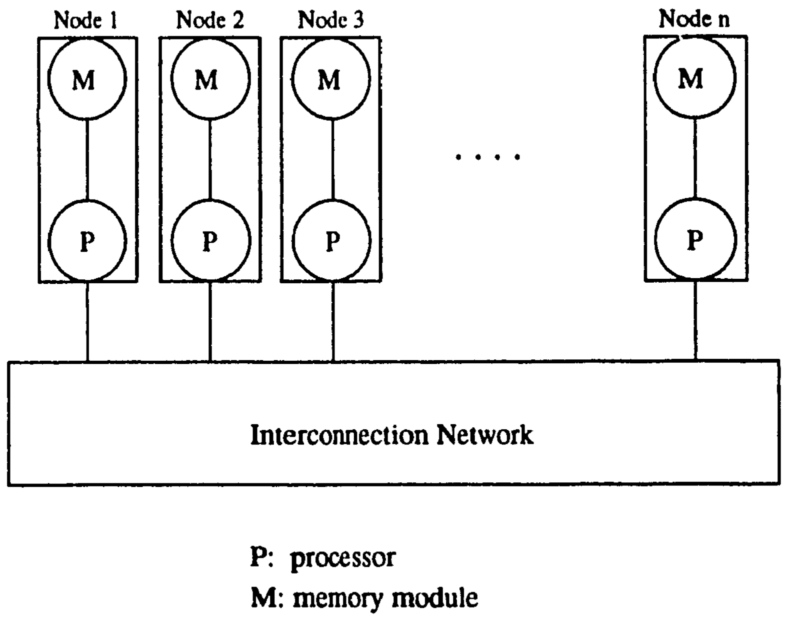


Figure 2.1: Structure of a multicomputer

When jobs are running on a MIMD machine, distinct jobs can be executed on different processors concurrently. A single job can be divided into a set of interactive processes, which also can be executed concurrently. The running time of this system of computations is thus improved. The abstraction of its computation model is depicted

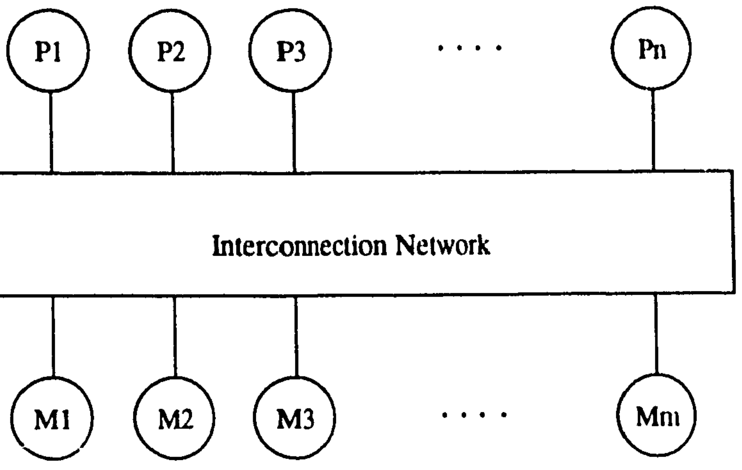


Figure 2.2: Structure of a multiprocessor

in Figure 2.3.

Uniprocessor systems have no fault tolerance because if the processor fails, the whole system fails. On the contrary, MIMD machines can have better fault tolerance. The processors in a MIMD machine are connected by the underlying network. If the network is properly designed, the machine can continue its execution, when a fault occurs, by either reconfiguring the network to a smaller size of the same network or to another class of network.

An interconnection network can be extended to accommodate more processors and memory modules. Therefore, the performance of a MIMD machine can be upgraded by adding in more hardware resources with minimum changes in software.

Since MIMD machines have many advantages over other systems, lots of efforts have been made to construct MIMD systems and to improve their performance. Many MIMD machines are now commercially available or being experimented with. For example, N-cube/10, Intel iPSC, Ametek S/14 [20], Cray-2 [25], Cosmic Cube [39], Mosaic [7], Connection Machine [22], Mark III [33], and J-machine [14]. The number

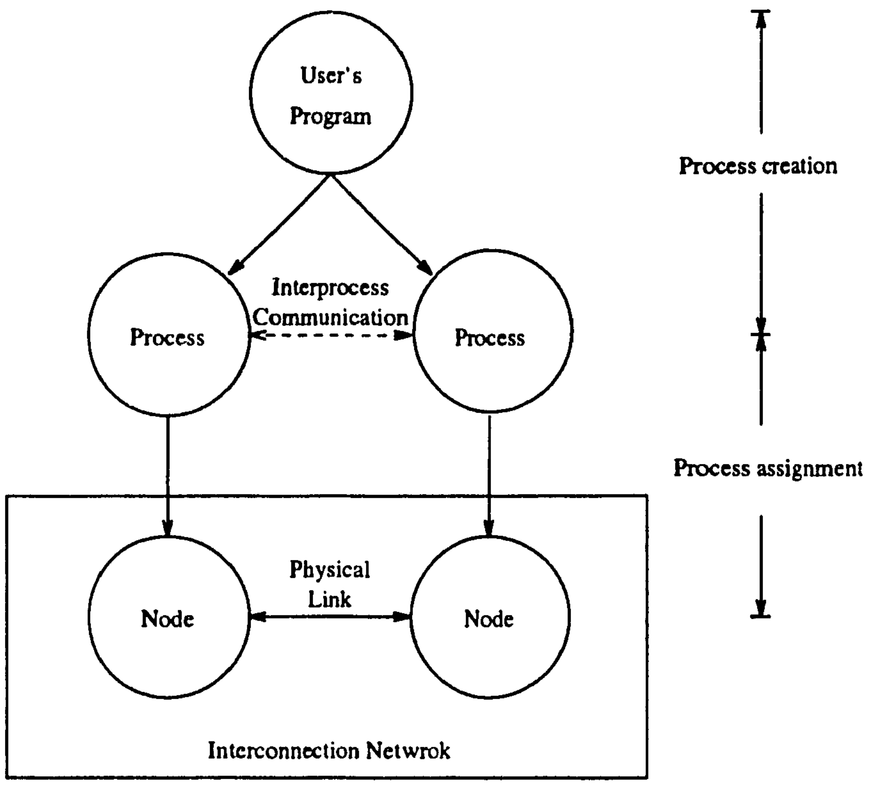


Figure 2.3: Model of computation in a MIMD machine

of processors ranges from 2 to 65536.

As described in the previous section, certain applications fit into the structures of MIMD machines. For example, to process an image, processors can be arranged in a manner such that each processor is dedicated to process a fixed pixel. Similarly, one processor is dedicated to process a concept in a semantic network, and one processor is used to represent a transistor in the simulation of VLSI. Many applications of these kinds demand the use of large-scale MIMD machines. However, an efficient large-scale MIMD machine needs a good coordination of operating system, architecture, and parallel algorithms. In particular, the architecture design is still a challenge in searching for efficient MIMD machines. For example, memory latency is the bottleneck of system performance in the conventional Von Neumann computation model. The speedup that can be made by adding more processors to a memory-sharing multiprocessor is not proportional to the number of processors and it tends to reach an upper bound [25]. This is because memory conflicts increase with the number of processors. By this argument, it is impractical to construct a large-scale multiprocessor or to scale its size. On the contrary, a multicomputer system has the advantage that it minimizes memory latency because each processor is tightly coupled to its memory. Every technique that was developed for minimizing memory latency, e.g., instruction-prefetching, is ready to be used on multicomputers.

Multicomputer systems can be further characterized as fine-grain and coarse-grain multicomputers according to the number of nodes or the size of each individual node. We refer the systems which contain tens or hundreds of nodes or each node has megabytes of memory as coarse-grain multicomputers. Those systems that contain thousands of nodes or each node has kilobytes of memory are classified as fine-grain multicomputers. A fine-grain multicomputer is expected to have better performance than a coarse-grain multicomputer because a fine-grain multicomputer allows even higher degrees of parallelism. With the rapid progress of VLSI technology, the power of a single chip is expected to outperform today's computers [25]. A multicomputer that contains thousands of single-chip nodes is both feasible and an attractive option for parallel processing [39]. However, it requires a better and more complex design of its interconnection network. The choice of interconnection network in a very large system is crucial as most of computations are done in the network. The design of a good interconnection network is very difficult because it involves lots of processors, it must accommodate some contradictory requirements, and people do not have much experience in designing large-scale networks [22].

2.3 Issues of Network Design

Computer networks are generally used to exchange data and to transfer files. Application programs are developed and built on top of a network to fulfill these tasks, as for example, electronic mail, file transfer protocol. Many networks have been developed for different environments. For example, the networks in a multicomputer and a multiprocessor are different. The Ethernet and pr[illegible]T for local area networks, the ARPANET for a wide area network [13], are all different. No matter what purpose the network is for, the design decisions of a computer network is made up of a 4-tuple, (topology, switching methodology, control strategy, operation mode) [18]. To have a complete view of computer networks, we briefly review these design issues in this section.

- Topology: Two classes of topologies can be identified, dynamic and static. In the dynamic topology, communication links can be reconfigured by switching elements to connect different processors. It is often used to connect processors and memory modules in multiprocessors. Examples of this topology are single-stage, multistage, and crossbar. In the static topology, each communication link is dedicated for two processors and provides end-to-end connection between them. There are no switching elements and links can not be reconfigured. It is often used to connect the processors in multicomputers, processing elements in array SIMD machines, and computers in a local area network. Examples of this topology are bus, ring, mesh.

- Switching methodology: Two major switching methodologies are used, circuit-switching and packet-switching. In circuit-switching, a dedicated physical path is allocated before transferring data. The dedicated path cannot be reused until the transmission is finished. It is efficient for transmission of bulk data. In packet-switching, there is no preallocated path for data transmission. Data are sent in packet-form and packets are routed separately in each individual node. A communication protocol is used to ensure that the correct sequence of packets will be received in the destination. It is efficient for transmission of short data. It is often used in computer networks. For example, the Ethernet and ARPANET, the popular local area and wide area networks, are packet-switching. An integrated-switching is a hybrid form of the above two switchings.

- Control strategy: There are centralized and distributed control. In centralized control, a central controller is used to regulate the network. The central controller can broadcast data or set the status of each switching element to alter connections. The networks in SIMD machines and multiprocessors are of this kind. In distributed control, there is no central controller. Each processor requests or stops communications based on its own decision. The networks in multicomputers, distributed systems, and computer networks belong to this category.

- Operation mode: Communication can be synchronous or asynchronous. In synchronous communication, the communication path is established synchronously. Both sender and receiver are ready for transmission. It is often used with centralized control in SIMD machines to broadcast data to processing elements from the control unit. In asynchronous mode, the communication path is established asynchronously. The sender requests communication dynamically and dose not wait for the receiver. The receiver may or may not wait if the data are not available. It is often used with the distributed control in multicomputers. The Ethernet is also asynchronous.

Since we only concentrate on the topology of a network, we will not explain the other issues in detail. We point out them for a complete view of computer networks.

The topology of a network can be static or dynamic. Our concern is large-scale computer networks, ranging in size from several hundreds to hundreds of thousands of nodes. Dynamic topology is not well adapted to large-scale networks because the number of switching elements is too large [22]. We therefore restrict our attention to static topology. In the next section, we will examine the performance issues of a computer network by using a multicomputer as a basis, since it takes more strict requirements on its network than any other systems.

2.4 Issues of Network Performance

An interconnection network provides end-to-end connections among all nodes in a multicomputer system. Each node runs a multiprogramming operating system kernel and executes its own processes with local data. Cooperation and interaction among the nodes is via message-send and message-receive instructions. These instructions cause messages to travel through the network from a source node to a destination node. For most cases, the communication channels are bidirectional, asynchronous, and full-duplex. A simple communication protocol is used to regulate messages. Each node can be divided into two sections: one section is for normal computation, the other section is for routing messages through the network, see Figure 2.4. The routing model is often based on the packet-switching model [29]. A message is broken into packets, which at least contain receiver's and sender's IDs, message type, message length, and message contents, see Figure 2.5. Only one packet can be sent along a channel at a time. A queue is thus generally used to temporarily store messages at each node. These messages include a node's own pending messages and the ones passing by from other nodes. A node picks up a message from the queue and sends to the next node in a pipeline fashion. The choice of next node is made by the routing algorithm, which can be stored at each node. Network latency thus arises and becomes an important issue in fine-grain multicomputers. The network latency is the sum of message delay time and the time to emit a message. It can be expressed as

$$Latency = T_p * D_t + L/B \tag{2.1}$$

T_p: the delay at each node

D_t: the number of nodes traversed

L : message length

B : channel bandwidth

However, the network latency is not the only factor that influences system performance. In fact, an interconnection network has crucial impact on the performance of a multicomputer. It is not only because there is a tradeoff among different interconnection strategies, but also because an inefficient network can cause protocol overrun or a corruption of the message-passing mechanism [42], which can make the whole system useless.

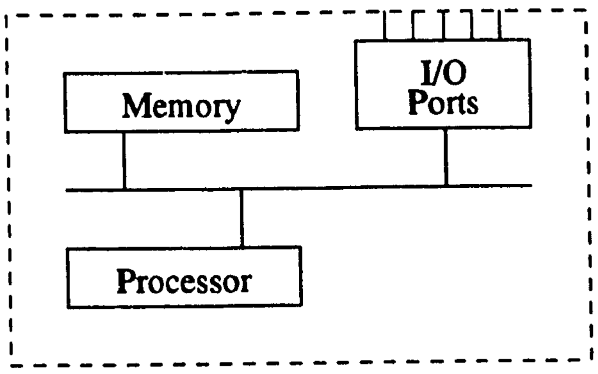


Figure 2.4: The structure of a node

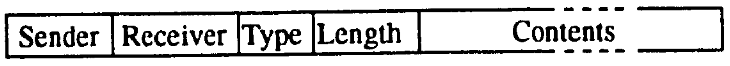


Figure 2.5: Example of a packet

The important issues on network performance are message delay, traffic congestion, ease of routing, fault tolerance, and extensibility [22, 8]. We discuss these issues below:

- message delay: It is the latency between the issuing of a message at the source and the receiving of the message at the destination and it corresponds the number of nodes traversed between the source and destination nodes.

- traffic congestion: Communication channels are limited to that only one packet can be sent at a time. Overloading any particular channel, the result is that messages have to wait in a queue. To avoid long waiting, the channels should be uniformly distributed. Traffic bottlenecks are then eliminated.

- ease of routing: If each node in a network has the same pattern of linkages, the same routing algorithm can be applied to each node in the network. The task of routing is thus simplified.

- fault tolerance: A good network should not crash when certain number of nodes or channels fail. Between any two nodes, there should be more than one path connecting them in order to survive failure of one or more nodes or channels.

- extensibility: A network should not have only one fixed size. It should be possible to extend it easily by allocating more nodes and thus to enhance its performance.

With these issue in mind, we must realize that these requirements contain contradictions. For example, communication wires cost money and a single chip has a limited number of connection pins. We cannot expect to lower latency by adding too many channels to a node. Similarly, we cannot expect to have many disjoint paths between two nodes when the number of channels per node is not high. Hence, compromises must be made among efficiency, reliability, and availability, based on the goal of an interconnection network.

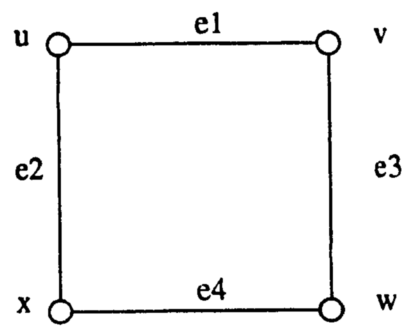
In addition, many parallel algorithms have been developed and applied on different networks for fast mathematical computations. We also should consider the possibility for a network to embed other networks, so the existing fast algorithms are ready to be applied on the network.

2.5 Basic Notions of Graph Theory

Before we relate the issues of network performance to graph-theoretic properties, we first give a review of some basic notions of graph theory that will be used in this thesis. Our introduction to graph theory is based on [19, 35].

A *simple graph* $G(V, E)$ is defined by a finite nonempty *vertex set* V and a finite *edge set* E of unordered pairs of distinct vertices from V. Each element in the set V is a vertex and each element in the set E is an edge of the graph G. There is a function γ from E to $V \times V$ and we write $\gamma(e) = (u, v)$ for an edge e and two vertices u and v if u and v are connected by the edge e. We say that vertices u and v are adjacent and that u (or v) is incident with the edge e. If two edges are incident with a common vertex, then they are adjacent edges.

From the definition, we know that a simple graph has the following properties: 1) no loops, any edge begins and ends at different vertices, 2) no multiple edges between any pair of vertices, 3) edges are undirected: edges can be traversed along either direction. In this thesis, we only consider simple graphs. An example of a graph is given in Figure 2.6.



G(V,E)
V={u,v,w,x}
E={(u,v),(u,x),(w,v),(w,x)}
f(e1)=(u,v), f(e2)=(u,x), f(e3)=(w,v), f(e4)=(w,x)
u and v are adjacent
e1 and e2 are adjacent
e3 and e4 are incident with w

Figure 2.6: Example of a graph

A graph $H(V_1, E_1)$ is a *subgraph* of $G(V, E)$ if and only if V_1 is a subset of V and E_1 is a subset of E and the function γ for G defined on E agrees with the function γ for H on E_1.

The *size* of a graph $G(V, E)$ is the number of vertices that the graph contains and is equal to $|V|$.

A *path* in a graph $G(V, E)$ is a sequence of edges such that the terminal vertex of an edge is the initial vertex of the next. Thus if $e_1, e_2, \ldots, e_n$ are in E, then $e_1, e_2, \ldots, e_n$ is a *path* provided there are vertices $x_1, x_2, \ldots, x_{n+1}$ in V so that $\gamma(e_i) = (x_i, x_{i+1})$ for $i = 1, \ldots, n$. The *length* of a path is the number of edges in the path.

The *distance* of two given vertices in a graph is the shortest path that connects them in the graph. The *diameter* of a graph is the largest distance among all pairs of vertices in the graph and is denoted by D.

A path with vertex sequence $x_1, x_2, \ldots, x_n, x_{n+1}$ is *closed* if $x_1 = x_{n+1}$. A *cycle* is a closed path in which $x_1, x_2, \ldots, x_n$ are all distinct.

A path with vertex sequence $x_1, x_2, x_3, \ldots, x_n$ is called a *Hamiltonian path* for a graph $G(V, E)$ if $x_1, x_2 \ldots, x_n$ are distinct and $\{x_1, x_2, \ldots, x_n\} = |V|$. A cycle $x_1, x_2, \ldots, x_n, x_1$ is a *Hamiltonian cycle* if $x_1, x_2, \ldots, x_n$ is a Hamiltonian path. A graph that has a Hamiltonian cycle is called a *Hamiltonian graph*.

The *degree* of a vertex is the number of edges that are incident with the vertex. The *degree of a graph* is the largest degree among all vertices in the graph and is denoted by Δ.

A graph is *regular* if every vertex in the graph has the same degree.

Two graphs G and H are *isomorphic* if there exists a one-to-one correspondence π on their vertex sets such that (u, v) is an edge of G if and only if $(\pi(u), \pi(v))$ is an edge of H. Such an π is called an *isomorphism* of G onto H. We call an isomorphism of a graph onto itself an *automorphism* of the graph.

A graph is *vertex-symmetric* if and only if for every pair of vertices, v_1 and v_2, there exists an automorphism of the graph that maps v_1 into v_2. Similarly, a graph is *edge-symmetric* if and only if for every pair of edges, e_1 and e_2, there exists an automorphism of the graph that maps e_1 into e_2.

The removal of a vertex u from a graph G results in the subgraph that contains all vertices except u and all edges not incident with u.

A graph is *connected* if given any two vertices in the graph there is a path connecting them.

The *connectivity* of a graph is the minimum number of vertices that need to be removed in order to disconnect the graph.

A family of graphs $\{G_1, G_2, \ldots, G_i, \ldots\}$ will be said to be *recursive* if G_i can be obtained from a number of copies of G_{i-1} by some simple operations and G_{i-1} is a subgraph of G_i. Adding a number of edges according to an algorithm is an example of the above simple operation and should be sufficient in general.

2.6 Design of Interconnection Networks

In designing an interconnection network, graphs are usually used to model the computer network in which the processors are represented by vertices and the communication links are represented by edges of a graph. The performance of an interconnection network modeled by such a graphical setting depends on the topology of the graph and can be analyzed by graph theory. Recall that the performance issues of a network are message delay, traffic congestion, ease of routing, fault-tolerance, and extensibility. We now address these issues in graph-theoretic terms and list their requirements below:

- message delay: The maximum message delay is proportional to the diameter of a graph. If the diameter is small, message traversal path is short. D_t is decreased in Formula 2.1 on page 15.
- traffic congestion: A regular graph has its edges uniformly distributed. Traffic bottleneck is eliminated. Queuing time at each node is short. T_p is decreased in Formula 2.1 on page 15. Furthermore, the task of node design is simplified because the same node structure can be used throughout the network.
- ease of routing: A vertex-symmetric graph has the properties that the graph looks the same at each vertex. The same routing algorithm can be applied at each vertex. Symmetry also implies regularity.
- fault tolerance: If a graph is highly connected, there are many disjoint paths between any two vertices in the graph. The ability of the network to survive from node or edge failure is enhanced.
- extensibility: If a graph has a recursive structure, it can be modified to varying sizes and retain the properties of the graph. This reconfiguration takes minimum changes in software because only tables and constants in resource managers need to be modified and application programs and compilers are preserved.

- links per node: It is equal to the degree of a vertex. It is constrained by the number of connection pins of a chip.

In summary, graphs that are used to model interconnection networks should have the following properties: *low diameter, regular, symmetric, highly connected, recursive, and of low degree*. We know that these properties can be contradictory from the previous section. So for past several years, researchers have been working on problems of constructing graphs based on different goals [12]. In particular, we point out two questions that are related to our work.

Problem 1. Given a D and a Δ, construct a graph of diameter D and degree Δ whose size is as large as possible.

Problem 2. Given a Δ and a S, construct a graph of degree Δ and size S whose diameter is as small as possible.

Problem 1 is the well-known (Δ,D) graph problem. Given a degree Δ and a diameter D, the size of a graph is bounded by the so-called *Moore bound*, which can be derived from the following argument:

The first node can link to at most Δ different nodes with diameter 1. Each Δ node at diameter 1 can link to Δ-1 different nodes with diameter 2. Each Δ(Δ-1) node can link to at most Δ-1 different nodes with diameter 3. By continuing this process, we reach vertices with diameter D and the graph is constructed in a tree shape. So an upper bound on the size of a graph with given (Δ,D) is a function of Δ and D. The function can be expressed as,

$$S(\Delta, D) = 1 + \Delta + \Delta(\Delta - 1) + \Delta(\Delta - 1)^2 + .. + \Delta(\Delta - 1)^{D-1} \tag{2.2}$$

From Formula 2.2, a lower bound on the diameter of a graph can be derived

$$D(S, \Delta) = \log_{\Delta-1} S - 2/\Delta \tag{2.3}$$

A graph of degree Δ and diameter D is called a *Moore graph* if its size is equal to S(Δ,D). The known Moore graphs are the following:

1. Complete graphs: D = 1, any Δ , S = 1+Δ

2. Rings: any D, $\Delta = 2$, $S = 2D + 1$
3. Petersen graph: $D = 2$, $\Delta = 3$, $S = 10$
4. Hoffman-Singleton graph: $D = 2$, $\Delta = 7$, $S = 50$

For $D = 2$ and $\Delta = 57$, the problem of its existence remains open. For the rest of the combinations, it is impossible to construct the corresponding Moore graphs [24] and it is not known how close we can get to the Moore bound. So far the random graphs are the best as far as the diameter is concerned. It is proved in [10] that almost all random regular graphs with degree Δ and size S have a diameter

$$D = log_{\Delta-1}S + log_{\Delta-1}logS + c \tag{2.4}$$

where c is a constant not greater than 10. This formula has the same order as Formula 2.3 and is very close the optimum. However, it is difficult to construct low-diameter random graphs by any explicit method [12] and it is also difficult to use random graphs to model interconnection networks because of the disadvantages of random graphs as stated in Section 1.2. Many strategies different from random graphs were developed to construct graphs of large sizes and low diameters.

As mentioned before, some properties of graphs are contradictory. It is impossible to construct graphs that include all good network properties. In this thesis, we propose a new construction method for large-scale interconnection networks. A larger latency will be exhibited when the network size increases. Methods have to be developed to lower the latency in large-scale networks. Therefore, our construction is based on this goal and is the type that is presented in Problem 2. This method provides a uniform construction for families of large graphs, which are called multidimensional linear congruential graphs. In Chapter 4, we will prove that our graphs are suitable for interconnection networks by those graph-theoretic properties discussed above. Since we derived those graph-theoretic properties from the interconnection network of a multicomputer, and any general-purpose computer network possesses many similarities to multicomputer systems, our model is suitable for multicomputers, computer networks, and distributed systems. We will also compare the size, diameter, and construction method of our graphs with some famous graphs in Chapter 5. The routing algorithm for this model is currently under investigation.

Chapter 3

Review of Current Network Models

3.1 Overview

In past years, many simple topologies were used to connect computers (processors), for example, array, ring, tree, mesh, and complete graph. These topologies have low cost and simple routing algorithms, however, none of them can be used for large-scale networks because their diameters are too large (except for complete graphs) and the degree of a large complete graph is too high. Many methods were developed to create new topologies or to modify existing ones, for example, the hypercube and its derivatives, cube-connected cycles, butterfly, shuffle-exchange, de Bruijn, Banyan, and Delta networks [29]. We summarize some important families of graphs that have good network properties as below.

1. Generalized chordal ring: chordal ring [4] and generalized chordal ring [17]
2. Graphs on symbols: de Bruijn, Kautz [9]
3. Product of graphs: optimal (3,3), (4,2) and (5,2) graphs [8]
4. Compound graphs: multitree (MTS) [5]

5. Graphs based on group theory: star, pancake [3]

In particular, we point out four topologies and examine their structures in the following sections.

3.2 The Hypercube Graph

3.2.1 Definition

The hypercube graph is also known as hypercube, boolean r-cube, binary r-cube, r-cube, cosmic cube. Its definition is given below.

DEFINITION 3.2.1 The r-dimensional Hypercube

In a r-dimensional hypercube graph, each vertex is represented by a r-bit binary string. Two vertices are connected by an edge if and only if their binary strings differ in exactly one bit.

An edge is called dimension k edge if it connects two vertices whose binary strings differ in the kth bit. Some examples of hypercubes are given in Figure 3.1.

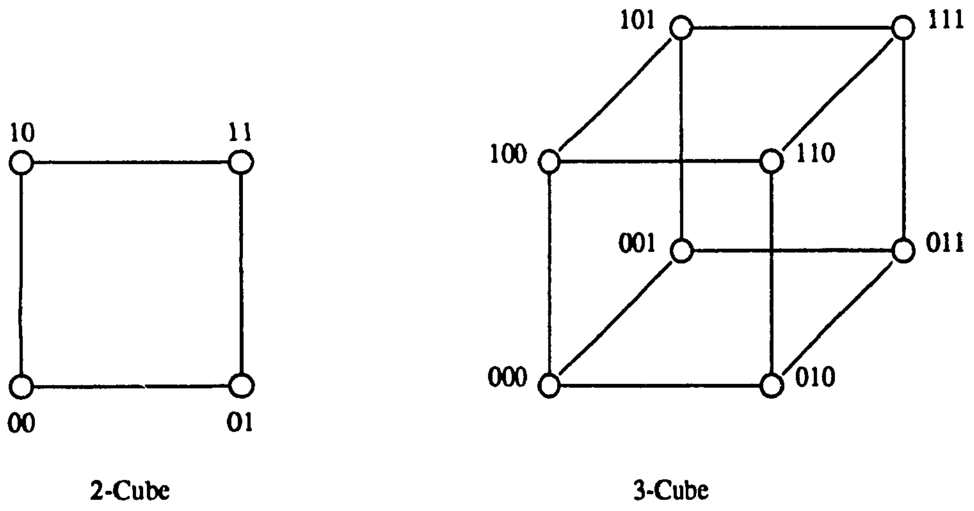


Figure 3.1: 2-cube and 3-cube

3.2.2 Properties of Hypercubes

The hypercube has many good network properties. Its structure, properties, and generalizations have been investigated by many researchers [29, 36, 38, 40]. Many algorithms were also developed to put it into real applications. It is one of the best understood networks. We review those properties that we pointed out in Chapter 2.

- Size: A r-dimensional hypercube has 2^r nodes since the number of combinations on a r-bit binary string is 2^r.
- Diameter: The diameter of this graph is r. The maximum distance to travel is between two vertices whose binary strings are complements of each other. This travel can be done by traversing r edges in which none of them lies in the same dimension.
- Regularity: It is regular. Each vertex has a degree r and therefore is connected to r vertices.
- Symmetry: It is both edge and vertex symmetric [29].
- Connectivity: It has the maximum connectivity. There are r vertex-disjoint paths between any pairs of vertices [36].
- Extensibility: Hypercubes have a recursive decomposition structure. A r-dimensional hypercube can be constructed from two copies of r-1 hypercubes by connecting their dimension r edges. An example is given in Figure 3.2.

3.2.3 Summary

The hypercube possesses many good network properties as seen in the previous subsection. Hypercubes can embed many well-known and popular networks efficiently, for example, a n-node hypercube contains one dimensional n-node and two dimensional $\sqrt{n} \times \sqrt{n}$ node arrays as subgraphs. It has been shown in [29] that the hypercube contains or nearly contains arbitrary arrays as subgraphs. It is also true for ring, binary

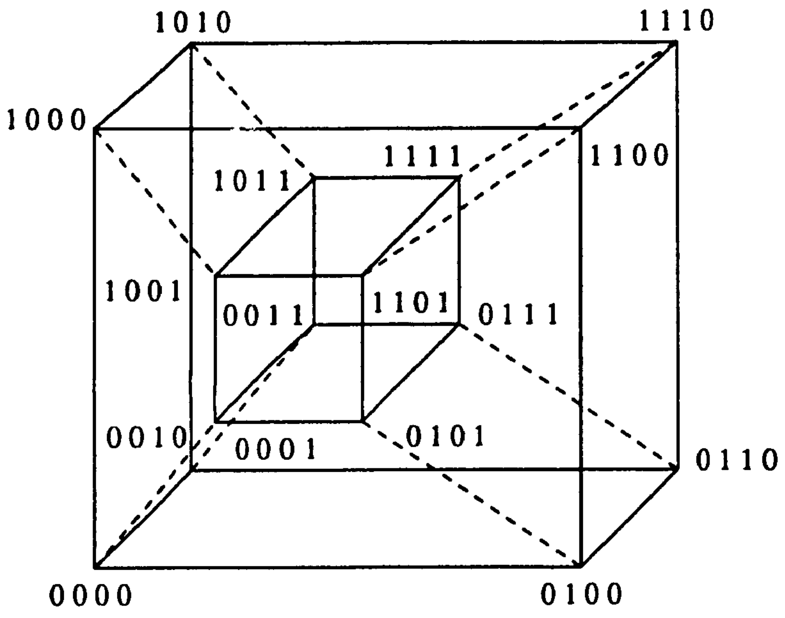


Figure 3.2: Two 3-cubes are linked to form a 4-cube by adding dashed edges

tree, mesh, and mesh of trees. Thus, the algorithms developed for these networks can be simulated by hypercubes with only a small factor slowdown. In addition, many parallel algorithms have been developed for the hypercube. For example, a $2^m \times 2^m$ matrix multiplication can be done by a 2m-cube in O(nlog$_2$ n) steps where $n = 2^m$ [25]. Its nice ructure and well-understood properties make it powerful and popular in the market. Many MIMD and SIMD array machines use hypercubes as their interconnection networks.

Examples of available multicomputers that employ hypercubes as their interconnection networks are Cosmic Cube [39], Intel iPSC/1, iPSC/2, Ametek S/14, N-Cube/10 [20], and Connection machine [22]. The dimensions of the cubes used in these systems range from 2 to 12.

However, the hypercube has three drawbacks:

1. Its size is restricted to powers of two. Variants can be found in [38, 40].

2. Its degree grows logarithmically with the network size because degree = log_2

size. It is impossible to increase the size without increasing its degree.

3. Its diameter equals its degree, so the diameter grows logarithmically with the network size. In fact, its diameter is the biggest one among the networks we discuss in this chapter.

Communication wires cost money and a single chip has a limited number of connection pins. When constructing a large-scale interconnection network, the cost of this many communication wires and the undesirability of a large latency make the hypercube a less favorable choice. Several bounded-degree networks have been designed to overcome the drawbacks of the hypercube, for example, the butterfly graph, shuffle-exchange graph, cube-connected cycle, and the de Bruijn graph. The de Bruijn graph is considered one of the best constructions for large-scale interconnection networks. We discuss it in the next section.

3.3 The de Bruijn Graph

3.3.1 Definition

The de Bruijn graphs were originally constructed as directed graphs and there are different versions of the definitions [9]. Here, we give a definition for the undirected case, which is similar to the definition of a hypercube.

DEFINITION 3.3.1 The de Bruijn Graph

Each vertex in a de Bruijn graph, B(d,D), is represented by a string of length D. Each symbol of the string is in {0,1,..,d-1}. The edges of a vertex can be classified as two sets, left-shift edges and right-shift edges. A vertex, $p_D p_{D-1} \cdots p_2 p_1$*, is connected to vertices with the form,* $x p_D \cdots p_2$ *by right-shift, and the form* $p_{D-1} \cdots p_1 x$ *by left-shift,* $x \in \{0,\ldots,d-1\}$.

From the above definition, the de Bruijn graph $B(d,D)$ is of degree $2d$. In fact, the left-shift edges correspond to out-degree edges and the right-shift edges are in-degree edges of a vertex in a directed graph. We only consider undirected graphs, so we view them as the same.

A r-dimensional binary de Bruijn graph is a special case of the above with $D = r$, and $d = 2$ and it is similar to the hypercube. A vertex is represented by a string $p_r p_{r-1} \cdots p_2 p_1$, where $p_r \in \{0,1\}$, and is connected to 4 neighbors with coordinates, $p_{r-1} p_{r-2} \cdots p_1 0$, $p_{r-1} p_{r-2} \cdots p_1 1$, $0 p_{r-1} \cdots p_2$, and $1 p_{r-1} \cdots p_2$. Some examples of binary de Bruijn graphs are shown in Figure 3.3.

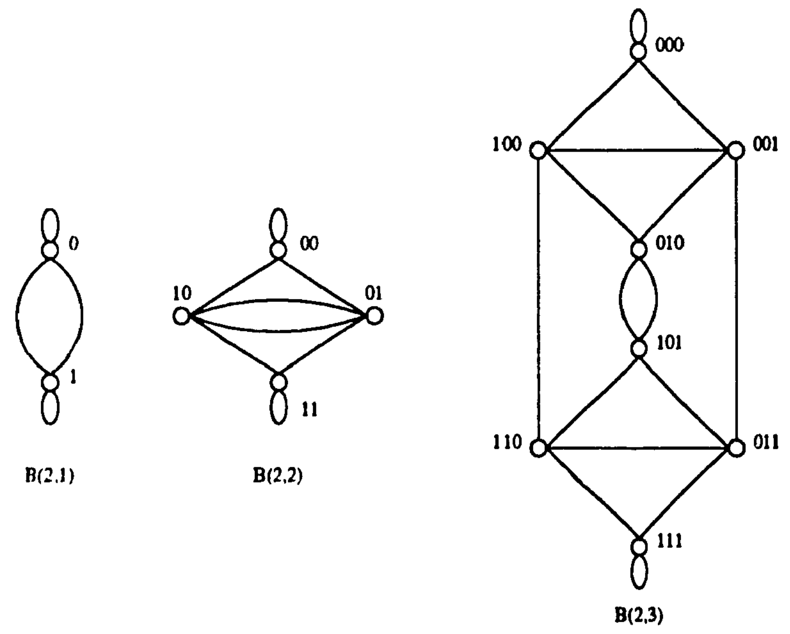


Figure 3.3: Examples of binary de Bruijn graphs

3.3.2 Properties of de Bruijn Graphs

The de Bruijn graph possesses good network properties in that it can solve wide classes of problems, e.g., pipeline class, multiplex class, etc, and it admits different sorting methods, e.g., sequential(parallel) input /sequential (parallel) output [37]. We first review the same basic properties as in the previous section.

- Size: The size of a de Bruijn graph, $B(d, D)$ is d^D because the combination of d symbols on length D is d^D , and is equal to 2^D for the binary graph.

- Diameter: The diameter is D. The maximum distance to travel is between two vertices whose strings contain no identical symbols. This travel can be done either by right-shift or left-shift D times from a source node.

- Regularity: The degree of a de Bruijn graph is $2d$ by its definition. The degree of a binary graph is therefore 4. However, the de Bruijn graph is not regular as for example the vertex $00\cdots0$ is only of degree $2d-2$.

- Symmetry: It is not fully symmetric. However, routing is very simple as implied by the definition of the graph. If a node, $x_1x_2\cdots x_D$, sends a message to another node, $y_1y_2\cdots y_D$, it can be done either by right- or left-shift D times. The route is thus a sequence, $x_1x_2\cdots x_D, x_2x_3\cdots y_1, x_3\cdots y_1y_2, \ldots, y_1y_2\cdots y_D$. Even though it is not the shortest path, one simple algorithm exists and is good for all nodes.

- Connectivity: Its connectivity is $2d-2$. It has the best possible connectivity in two senses. First, because the connectivity equals the minimum of the degrees of the graph, second, because it is possible to modify the graph slightly to obtain graphs which are regular and connectivity is $2d$ [40].

- Extensibility: A $B(d, D)$ graph can be constructed from either a $B(d-1, D)$ or a $B(d, D-1)$ graph. The difference of these two algorithms [29, 37] is that $B(d, D-1)$ is not a subgraph of $B(d, D)$, but $B(d-1, D)$ is a subgraph of $B(d, D)$. The algorithm to construct a $B(d, D)$ graph from a $B(d-1, D)$ takes d copies of $B(d-1, D)$ and is more complex than that of a hypercube. An example is given in Figure 3.4.

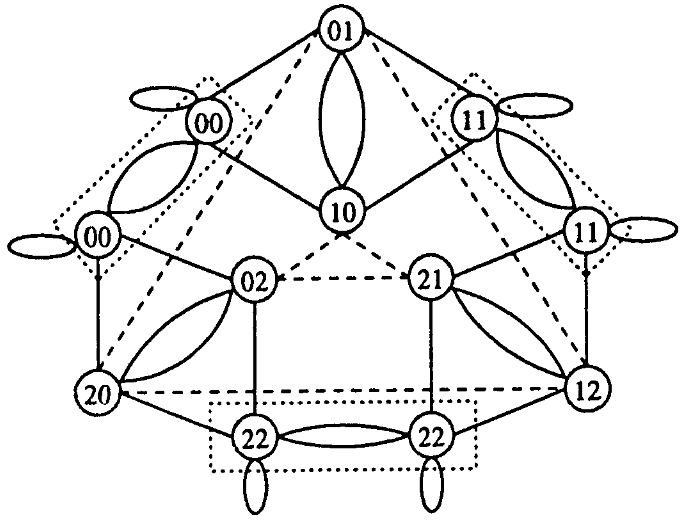


Figure 3.4: 3 copies of B(2,2) are linked to form a B(3,2) by dashed edges and combining nodes in dashed rectangles

3.3.3 Summary

The de Bruijn graphs are rich in structure and have most of the good network properties of hypercubes. They are easy for routing, optimally connected, extensible, and contain many more vertices than the hypercube for degrees greater than 4. Even though the binary de Bruijn graph and the hypercube have the same size and diameter, the binary de Bruijn graph still has the advantage that its degree remains at 4 as its size grows, and therefore the number of edges is much less than that of a hypercube for dimension greater than 4. Furthermore, the degree and diameter are not related for de Bruijn graphs. The advantage is that, given a size, there are many combinations of degrees and diameters for it, while it is unique for the hypercube. For example, given a size 1024, the only configuration of hypercube is degree 10 and diameter 10. The configurations of de Bruijn graphs can be B(2,10), which is degree 4 and diameter 10, or B(4,5), which is degree 8 and diameter 5. Similar to the hypercube, the de Bruijn graph can embed ring, linear array, and tree nicely. It also can embed those networks that are impossible or difficult for the hypercube, for example, tree machine, butterfly, shuffle-exchange graph. The binary de Bruijn graph can simulate those algorithms developed for the hypercube efficiently. An n-node de Bruijn graph can simulate an n-node hypercube algorithm without loss of efficiency [29]. Its ability for sorting and optimal VLSI layout had been justified in [37]. One limitation of this graph is that it is defined only for even degree. Although there is no commercially available multicomputer that uses the de Bruijn graph as its interconnection network, a special-purpose machine, which is used for a decoding of satellite communication in NASA's Galileo mission, employed the de Bruijn graph to connect its 8192 nodes [16]. From the point of view of large-scale interconnection networks, the de Bruijn graph is clearly much superior to the hypercube and can be a candidate for fine-grain multicomputers of the next generation.

The hypercube and de Bruijn graph could be considered to be in the same class: they are graphs that are generated on alphabets, and their diameters grow logarithmically with graph sizes. Given a diameter D and a size S, $D = \log_d S$ is the formula for the diameter of a de Bruijn graph $B(d, D)$. A new class of construction is introduced in the next section.

3.4 The Cayley Graph

3.4.1 Definition

The Cayley graph is based on group theory. Given a finite group G and a set of generators F, the vertices of a Cayley graph are the elements of the group G. There is an edge between vertices x and y if and only if there is a generator g in F such that $xg = y$. Following Akers and Krishnamurthy [3, 1], we give the definition of Cayley graphs that are constructed by permutation groups.

DEFINITION 3.4.1 Cayley Graphs on Permutations

Given a set of n symbols and a set of generators $F = \{g_1, g_2, \ldots, g_i\}$ where n and i are some integers, and each generator in F is a permutation, the vertices of the graph are all permutations on the n symbols. There is an edge between vertices x and y if *and only if for some* g_j, $1 \leq j \leq i$, $xg_j = y$.

The set of generators is required to satisfy the following: 1) it does not contain the identity permutation, 2) all generators are different, and 3) it is closed under inverse. With these requirements, the resulting graph has the properties: 1) there is no loops, 2) there is no no multiple edges between any pair of vertices, and 3) it can be viewed as undirected. The reason of the third propertiy is because for any two vertices x and y, if there is a generator g such that $xg = y$, then there must be a generator h in F such that $yh = x$. A Cayley graph is completely specified by its permutations and we will use n to denote the length of a permutation. We give some examples of Cayley graphs in Figure 3.5 and Figure 3.6.

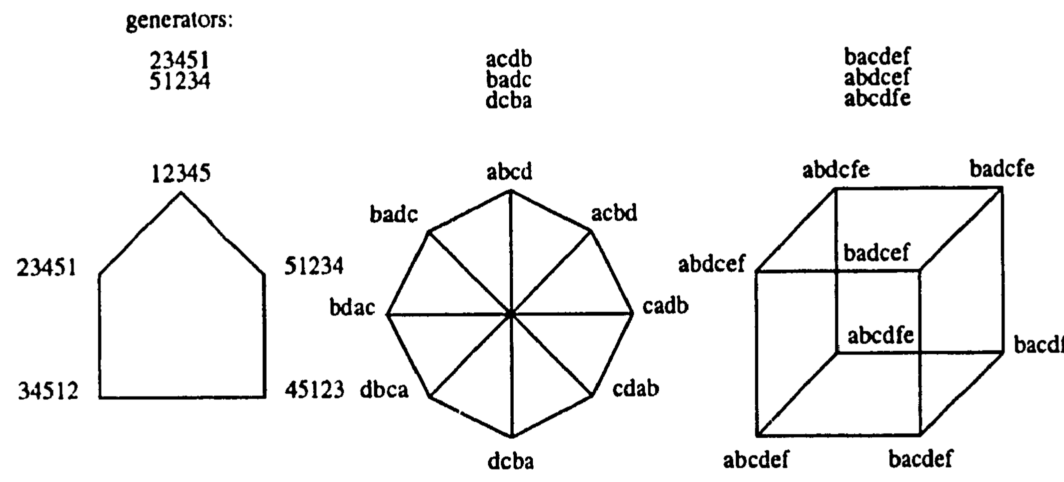


Figure 3.5: Examples of Cayley graphs

3.4.2 Properties of Permutation Cayley Graphs

As can be seen from Figure 3.5 and Figure 3.6, the Cayley graph can be used to create a variety of graphs based on the chosen permutations. For example, star graph, pancake graph, bubble sort graph have been considered [1]. In particular, we will use the star graph as an example to check its diameter and those graph-theoretic properties mentioned before.

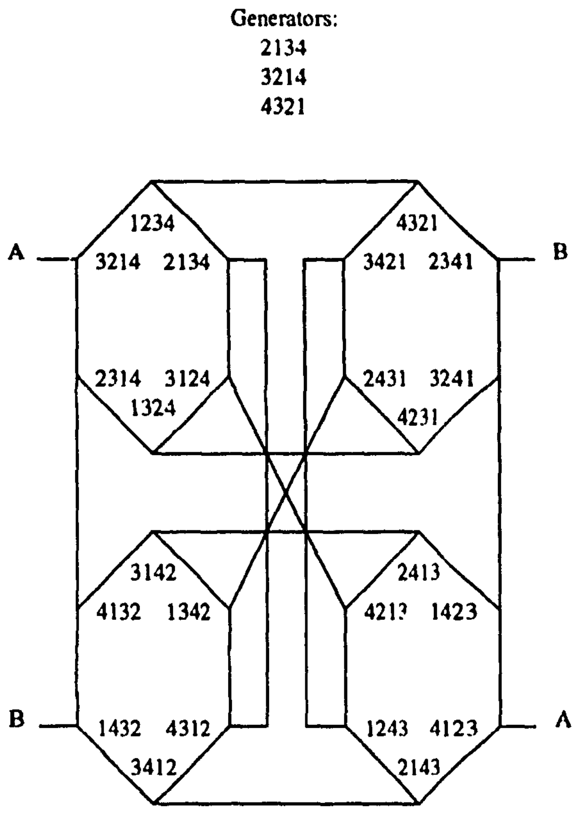


Figure 3.6: A 4-pancake graph

A n-star graph is a Cayley graph on n symbols and edges are constructed by $n-1$ generators in which each generator g_i, $2 \leq i \leq n$ transposes the first symbol with the ith symbol at each vertex. For example, a 3-star graph is given in Figure 3.7.

Now, let us look at the properties of permutation Cayley graphs.

- Size: The size of a graph is $n!$ since the number of permutations on n symbols is $n!$. Depending on the choice of generators, the $n!$ vertices may not all be connected.

- Diameter: Diameters of some graphs are still open. For those graphs with known diameters, the results can be found in [1, 2]. The diameter of a n-star graph is $\lfloor 3/2(n-1) \rfloor$.

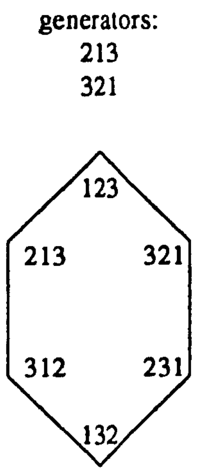


Figure 3.7: A 3-star graph

- Regularity: Since each generator is applied to a vertex once and the set of generators is closed under inverse, the Cayley graphs are regular and the number of generators is equal to its degree.
- Symmetry: It has been proved that every Cayley graph is vertex symmetric. It is also possible to be edge symmetric if the generators satisfy a specific rule [3]. A star graph is both vertex and edge symmetric. Routing is simple in the sense that finding a path between two vertices x and y is reduced to the sorting $y^{-1}x$ to I, the identity permutation.
- Connectivity: A n-star graph has the maximum connectivity $n - 1$ [2].
- Extensibility: A n-star graph can be constructed by n copies of n-1 star graphs. This is shown in Figure 3.8.

3.4.3 Summary

The Cayley graph is a very general construction. Many symmetric networks can be represented by these graphs. Examples are the cube-connected cycle, hypercube,

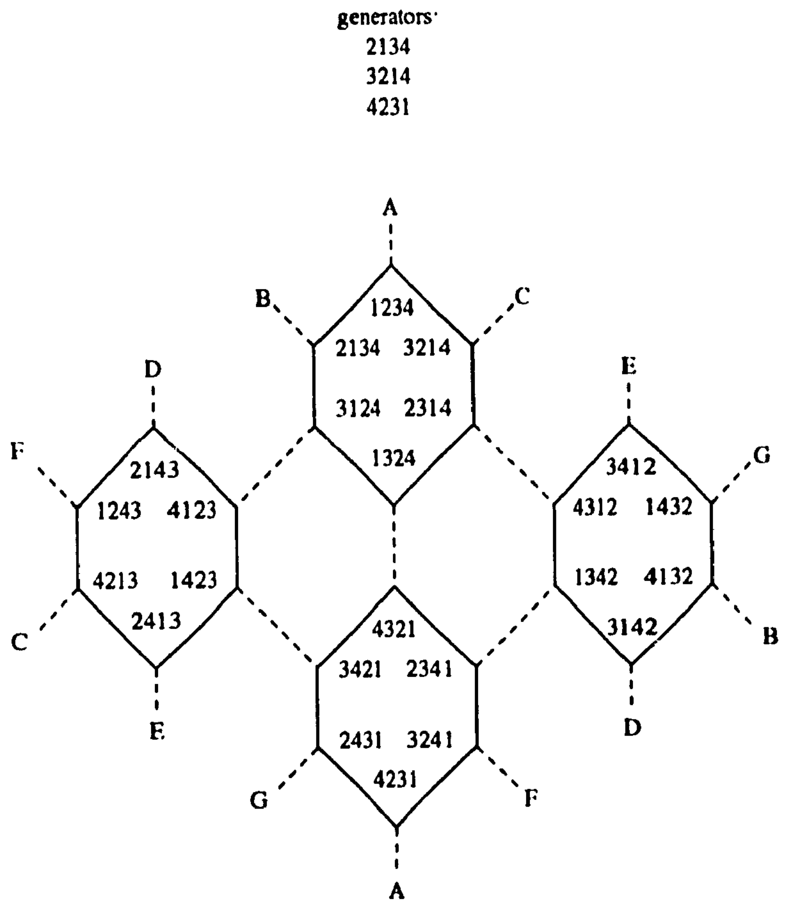


Figure 3.8: Four 3-star graphs are linked to form a 4-star graph by dashed edges

ring. Every symmetric network can also be constructed by this graph with a simple extension. Examples are the n-dimensional cube-connected cycle, and the burnt pancake graph. Moreover, it can be used to create new graphs by starting with an arbitrary finite group. The properties of a permutation Cayley graph are completely specified by n and the set of permutations. The star graph in the previous section, for example, has many properties to be a good interconnection network. It is symmetric, maximally connected, recursive, and its diameter and degree grows slower than the hypercube.

Other advantages of Cayley graph can be realized from the following:

- It provides algebraic tools for design and analysis.
- The properties of Cayley graphs can be proved as a whole class, instead of proving a property for each individual graph.
- All new created graphs inherit the known properties of Cayley graphs. For example, the property of vertex symmetry.

Thus, the Cayley graph provides an efficient way to design and analyze interconnection networks. However, like the hypercube, its drawback is that its degree grows with the graph size.

3.5 The DCC Linear Congruential Graph

3.5.1 Definition

The Disjoint Consecutive Cycles (DCC for short) linear congruential graph was proposed by Opatrny and Sotteau in 1992 [31]. Its construction shares the same idea as the Cayley graph since they both use the concept of generators. Its structure is as flexible as the de Bruijn graph since the degree is not predetermined by the chosen graph size. We first introduce the definition of a linear congruential graph.

DEFINITION 3.5.1 The Linear Congruential Graph

In a linear congruential graph of size n, each vertex is represented by an integer x, $0 \leq x \leq n-1$. *There is a set of generators* $F = \{g_0, g_1, \ldots, g_{i-1}\}$ *in which each one is a linear function. Two vertices x and y in a linear congruential graph* $G(F, n)$ *are connected by an edge if and only if there is a generator* g_j, $0 \leq j \leq i-1$ *such that* $y = g_j(x) \bmod n$.

The generator set in a Cayley graph is closed under inverse but it is not required to be so in a linear congruential graph. Hence, each generator adds an in-degree edge and an out-degree edge for each vertex. Since we only consider undirected graphs, it is assumed that all edges are bidirectional.

The disjoint consecutive cycles created by a generator set $F = \{g_0, g_1, \ldots, g_{i-1}\}$ on a graph are defined as follow.

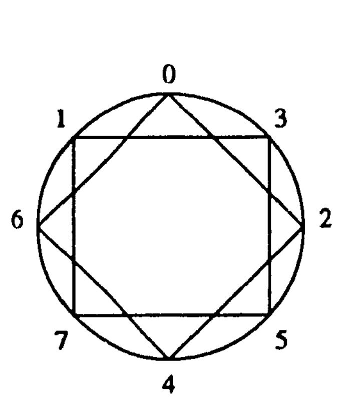
DEFINITION 3.5.2 Disjoint Consecutive Cycles

Given a set of generators $F = \{g_0, g_1, \ldots, g_{i-1}\}$ *for a linear congruential graph of size* $n = k^l m$ *for some integers* $k \geq 1$, $l \geq 2$, *and m. The cycles created by the generator set F in the graph are called disjoint consecutive cycles if each* g_j, $0 \leq j \leq i-1$, *creates* k^j *vertex-disjoint cycles of length* n/k^j *on the vertex set* $\{0, 1, \ldots, n-1\}$.

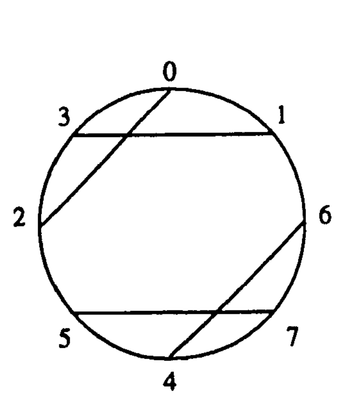
DEFINITION 3.5.3 The DCC Linear Congruential Graph

A DCC linear congruential graph is a linear congruential graph whose generators generate disjoint consecutive cycles (DCC) on the vertex set.

Because each generator adds 2 more degrees to a vertex, $d/2$ generators are used for a graph of even degree d. For a graph of odd-degree $d+1$, $d/2$ generators are used in the same way as the graph of degree d and an extra linear function g is used to generate an edge for every two vertices. An odd-degree graph is represented by $G(F, n, g, V_1)$ where V_1, which is a subset of the set $\{0, 1, \ldots, n-1\}$, is the domain for g. Examples of linear congruential graphs are given in Figure 3.9



G({5x+1,9x+2},8)



G({5x+1},8,{17x+2},{0,4,1,5})

Figure 3.9: Examples of linear congruential graphs

3.5.2 Properties of DCC Linear Congruential Graphs

Similar to the Cayley graph, the linear congruential graph can be used to construct a variety of graphs and the linkages are completely specified by the set of generators. However, the most distinguishing property of the linear congruential graph is its size. Its graph size is one of the input parameters for construction, while all previous models have other input parameters and their sizes are implied by these parameters. This gives linear congruential graphs a more flexible structure than the de Bruijn graph and lower diameters may be achieved by varying generators and graph sizes. Following the known results in [31] , we review the DCC linear congruential graphs with size formed by powers of 2.

- Size: The graph size is one of input parameters for construction. It is thus possible to construct variable-size graphs with any degree. The results showed that DCC linear congruential graphs are much larger than all the previous graphs of the same degree and diameter. We will have a detailed comparison in Chapter 5.

- Diameter: The problem of obtaining a formula for the diameters of these graphs is still open. There is an upper bound for the graphs of degree 4 but it is not very precise. Therefore, the diameters of these graphs must be calculated by computers. These calculated diameters show that the diameters of DCC linear congruential graphs are much smaller than the diameters of other families of graphs. For example, the Mosaic C, a multicomputer with 16384 nodes, used a 3-dimensional mesh as its interconnection network. Its diameter is thus 78. The diameter of the DCC linear congruential graph of the same size and degree is only 17.

- Regularity: It is regular since the generators partition the vertex set into different subsets and each generator is used to generate an edge for a vertex.
- Symmetry: It is not symmetric in most cases.
- Connectivity: Given a set F of DCC linear functions of size t, the graph generated by F is $2t$-connected. That is, an even-degree graph has the maximum connectivity. However, it is not necessarily $2t+1$-connected for an odd-degree graph, but it is at least $2t$-connected.
- Extensibility: DCC linear congruential graphs of size 2^{i} have a recursive structure. A graph of size 2^{i} can be constructed by two copies of graphs of size 2^{i-1}. An example is given in Figure 3.10.

3.5.3 Summary

Like de Bruijn graphs, the linear congruential graphs provide a uniform method to construct large-scale interconnection networks. Following the same thrust as Cayley graphs, linear congruential graphs use the idea of generators, and thus many known graphs can be represented by this graph with a proper generator set. Properties of these graphs can be proved as a whole class by algebraic tools. The superiority of linear congruential graphs over the other two graphs can be seen from the following:

1. It constructs graphs with larger sizes (relatively, lower diameter)

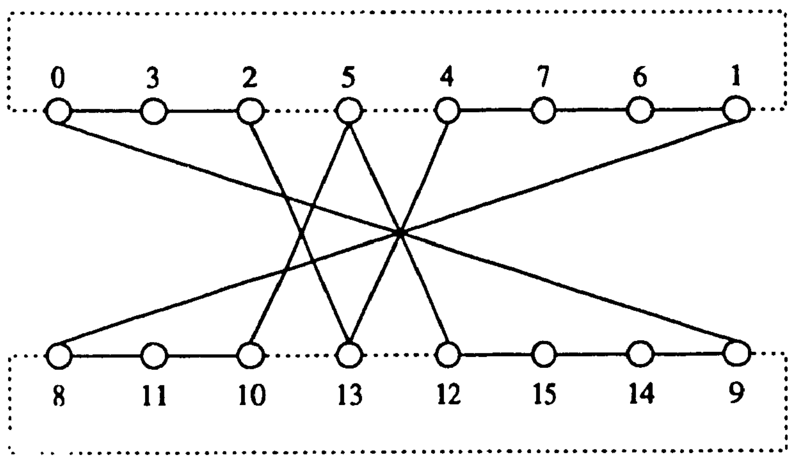

0
3
2
5
4
7
6
1
8
11
10
13
12
15
14
9
Dashed edges are in the original graph and deleted when it is extended.

Figure 3.10: Two copies of G({5x+3},8) are linked to form a G({5x+3},16)

2. Its degree is independent of its size.

3. Its diameter grows at a lower rate as a function of graph size than the other families of graphs.

4. It provides more varieties of choices to network designers because given a graph size, the linear congruential graph can construct graphs of any degree with that size. But it is not possible for the de Bruijn graph, for example, given a graph of size 512, the possible configurations are B(2,9), which is degree 4, and B(8,3), which is degree 16. The Cayley graph is more restricted because a n-Cayley graph has a size n! and degree n-1.

5. It generalizes the construction of de Bruijn graphs.

Therefore, the DCC linear congruential graphs have many advantages and can be considered as an alternative for the design of large-scale interconnection networks. However, it is still open to obtain a better bound on the diameter of a DCC linear congruential graph, and the precise relation among diameter, generators, and graph size has not not been found yet. In the next chapter, a new construction method is proposed. It generalizes the construction of linear congruential graphs. Our goal is thus to improve the diameters of DCC linear congruential graphs by constructing graphs with this new method, and to retain in these new graphs those good network properties found in the DCC linear congruential graphs.

Chapter 4

Multidimensional Linear Congruential Graphs

4.1 Overview

To have a better understanding of our multidimensional linear congruential graphs, we describe their structure informally before we come to a formal definition.

Let n be an integer, the number of dimensions of the graph. The size of an n-dimensional space used by our construction is bounded by a given vector of integers $(s_1, s_2, \ldots, s_n)$ in which each component s_i, $1 \leq i \leq n$, is the length of the ith dimension. The vertices of this graph are points in the n-dimensional space. Thus each vertex is assigned a vector of integers $(p_1, p_2, \ldots, p_n)$ in which each component p_i, $1 \leq i \leq n$, is in $\{0, 1, \ldots, s_i - 1\}$. Therefore, the size of the graph is $s_1 * s_2 * \cdots * s_n$. A vertex (or a vector) $(p_1, p_2, \ldots, p_n)$ will be represented by vector $\vec{p}$ in some cases. The former is used when the coordinates (components) of a vertex (a vector) are to be differentiated; the latter is used when the coordinates (components) can be viewed as a single unit. We choose vector representation to illustrate that the vertex of this graph is different from the vertex of a linear congruential graph, which is a single integer. The edges of this graph are defined by a set of n-dimensional linear functions $\{f_1, f_2, \ldots, f_k\}$ for some integer k. An n-dimensional linear function $\vec{x}A + \vec{b}$ consists of a n by n matrix A and a 1 by n vector $\vec{b}$. Given a vertex $(p_1, p_2, \ldots, p_n)$, it is linked to the vertex $(q_1, q_2, \ldots, q_n)$ by the generator $f_i(\vec{x}) = \vec{x}A_i + \vec{b_i}$ if the following formula holds,

$$(q_1, \ldots, q_n) = (p_1, \ldots, p_n) \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{n1} & \ldots & a_{nn} \end{pmatrix} + (b_1, \ldots, b_n) \bmod (s_1, \ldots, s_n)$$

$$\text{where } A_i = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \text{ and } \vec{b_i} = (b_1, b_2, \ldots, b_n).$$

As we can see, a linear congruential graph is a special case of this model when $n = 1$. The formula is simplied to $q_1 = (p_1 * a_{11} + b_1) \bmod s_1$ and each vertex is an integer. The idea of such edge-generation is illustrated in Figure 4.1 for the two-dimensional case.

To construct an odd-degree graph, we use an additional linear function to define a perfect matching on the vertex set. That is, an edge will be generated for every two vertices by this linear function.

We now give some informal reasons, which made us believe that this model can give good results as far as the diameter is concerned. First, let us look at an extreme example, the hypercube. The hypercube is good because of its simple and regular structure, as was shown in the figures of 3- and 4-cubes in Chapter 3. If we put the hypercube in the Cartesian coordinate axes, a vertex can be connected to other vertices only by edges that are parallel to the axes. By this constraint, the diameter of a 3-dimensional cube, for example, is 3. If we are allowed to construct those edges that are not parallel to the axes, we can easily get a graph of the same size and its diameter is 2. This is illustrated in Figure 4.2.

This example reminds us of the idea that a regular graph with a simple construction rule can be more easily understood but the diameter of the graph is not necessarily good. Among the graphs we mentioned before, the hypercube has the simplest structure, and is the easiest one to understand, but is the worst one when

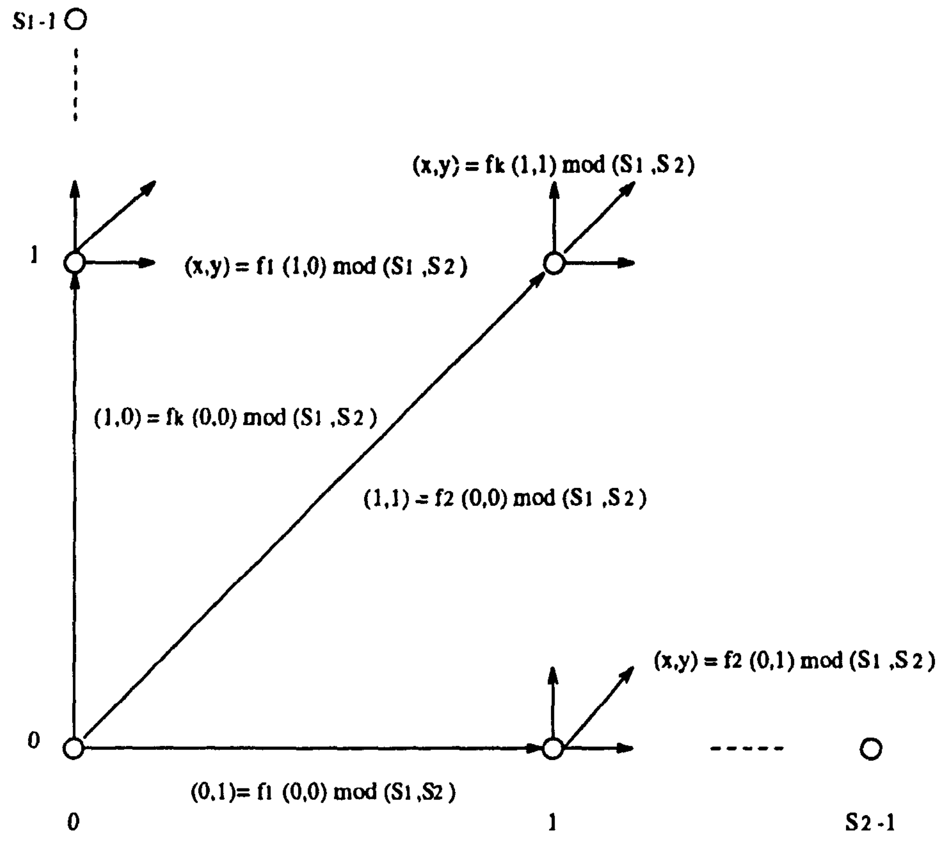

S1-1
1
0
0
1
S2-1
(x,y) = fk (1,1) mod (S1 ,S2)
(x,y) = f1 (1,0) mod (S1 ,S2)
(1,0) = fk (0,0) mod (S1 ,S2)
(1,1) = f2 (0,0) mod (S1 ,S2)
(x,y) = f2 (0,1) mod (S1 ,S2)
(0,1)= f1 (0,0) mod (S1,S2)

Figure 4.1: Example of a two-dimensional graph

considering its large diameter.

Second, given a linear function $f(x) = ax+c$, $a, c \in N$, and an integer x_0, $0 \leq x_0 \leq n-1$ for some integer n, a linear congruential sequence [28] $x_0, x_1, x_2, \cdots, x_j$ is defined by $x_j = f(x_{j-1})$ mod n. It is a periodic sequence of period equal to or less than n. This property was used in the linear congruential graph described in Section 3.5. Each x_j in the above sequence is a vertex of a linear congruential graph and n is the graph size. Since a linear congruential sequence is a periodic sequence, the linear congruential sequences created by $f(x)$ will define a number of cycles in a linear congruential graph $G(\{f\}, n)$. The linear congruential sequence of length n is thus a Hamiltonian cycle in the graph. The use of linear functions means that a de Bruijn graph is a special case of a linear congruential graph. The de Bruijn graph $B(d, D)$ is isomorphic to the linear

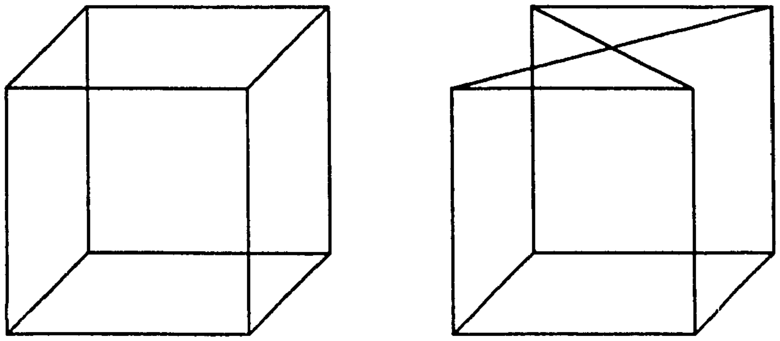


Figure 4.2: The 8-node 3-cube of diameter 3 and a 8-node graph of diameter 2

congruential graph $G(F, d^D)$ where $F = \{dx + i | 0 \leq i \leq d - 1\}$. Observing the linear functions in the set F, the same multiplicative constant is used for all linear functions. It seems natural to consider linear functions in which all multiplicative and additive constants are different. Inspired by the first reasoning, these linear functions should define edge sets that give more possibilities of linkages, and graphs constructed by these functions are expected to have smaller diameters. Since random graphs almost always have low diameters and linear congruential sequences with maximum length n are used to generate pseudo-random numbers [28], a linear congruential graph that contains one Hamiltonian and several shorter cycles is expected to have a low diameter [31]. Indeed, the diameter of a DCC linear congruential graph is smaller than the diameter of a de Bruijn graph.

Third, although it is impossible to describe the structure of a random graph by a small number of functions, the reason we expect this model to produce better results is that we are establishing an even more complex construction rule than the one used in a linear congruential graph. Inspired by the second reasoning, we expect that this complex rule will provide more possibilities of linkages and therefore lower diameters could be obtained. In a linear congruential graph, a linear function contains exactly one variable. As a result, the linear function is fixed for all vertices and is used to generate edges throughout the vertex set. However, as edges are created by a linear function in a multidimensional graph, the linear function can vary along its vertex set if it contains more than one variable. Edges are therefore actually created by two or more of those linear functions used in a one-dimensional linear congruential graph.

In general, the advantages of graphs in a multidimensional space over graphs on a linear axis are:

1. Vertices are divided into classes according to the dimensions and each class has its own linear function. As edges are generated along the vertices, a function varies in a way such that different functions are used for source vertices that have different classes.

2. The average distance of jumps can be expected to be farther because as an edge travels from one dimension to another, vertices between the two dimensions are all skipped. Such jumps can be made by switching dimensions and this can be done easier than the linear case because of the existence of extra variables.

Therefore, with more linear functions among vertices and a proper mixture of long and short jumps, we expect better diameters to be found. Indeed, it will be shown later that the size of a two-dimensional graph is larger than the DCC linear congruential graph of the same degree and diameter. Comparisons will be given in Chapter 5.

4.2 Definition

We will use N to denote the set of nonnegative integers and Z_p to denote the integer set $\{0,1,\ldots,p-1\}$. All elements of any vector or matrix are in N.

DEFINITION 4.2.1 Linear Functions of Dimension d

Let $d \in N$. We say the function f is a linear function of dimension d if $f(\vec{x}) = \vec{x}A + \vec{b}$ where A is a d by d constant matrix, $\vec{b}$ is a constant vector of length d, and $\vec{x}$ is a variable vector of length d.

A d-dimensional linear function in the above definition has d variables by its definition and can be rewritten to d linear functions based on matrix operations, one linear function for each dimension. Each of these rewritten functions, however, has at least one and at most d variables.

DEFINITION 4.2.2 Multidimensional Linear Congruential Graphs

Let $\vec{s}$ be a constant vector of length d, $\vec{s} = (s_1, s_2, \ldots, s_d), s_i \in N - \{0\}$ for $1 \leq i \leq d$, and F be the set of linear functions of dimension d, $F = \{ f_i(\vec{x}) | f_i(\vec{x}) = \vec{x}A_i + \vec{b_i}$ where $1 \leq i \leq k$, for some k $\}$. We define a graph $G(F,\vec{s})$ of dimension d as a graph on the vertex set $V = Z_{s_1} \times Z_{s_2} \times \cdots \times Z_{s_d}$, in which any $\vec{x} \in V$ is adjacent to the vertices $f_i(\vec{x}) \bmod \vec{s}, 1 \leq i \leq k$. For a subset V_1 of V and a linear function g, we define a graph $G(F,\vec{s},g,V_1)$ of dimension d as a graph on the vertex set V, in which any $\vec{x} \in V$ is adjacent to the vertices $f_i(\vec{x}) \bmod \vec{s}, 1 \leq i \leq k$ and any $\vec{x} \in V_1$ is also adjacent to the vertex $g(\vec{x}) \bmod \vec{s}$.

We use $G(F,\vec{s})$ to generate large regular graphs of even degree, while $G(F,\vec{s},g,V_1)$ will be used to generate large regular graphs of odd degree. The size of the above graph is given by $\vec{s}$ and is equal to $s_1 * s_2 * \cdots * s_d$.

We will call the linear functions in F and $F \cup g$, the generators of $G(F,\vec{s})$ and $G(F,\vec{s},g,V_1)$, respectively. For any generator f we will call the graph $G(\{f\},\vec{s})$ the graph generated by f on $\vec{s}$. Furthermore, a generator can be rewritten into d linear functions based on matrix operations, one for each component of a vector.

It is natural to consider that direct edges are from any $\vec{x} \in V$ to the vertices $f_i(\vec{x}) \bmod \vec{s}, 1 \leq i \leq k$. We restrict this thesis to the undirected case.

Linear congruential graphs used two important properties of linear functions, we restate them below since they will be needed in some of our results. Proofs can be found in [28] and [31].

LEMMA 4.2.1 Linear Congruential Sequences with Maximum Period

Let $f(x) = ax+c$ be a linear function, n be a positive integer and $x \in \{0, 1, \ldots, n-1\}$. The linear congruential sequence $x_0, x_1, \ldots, x_j$, defined by $x_j = (ax_{j-1}+c) \bmod n$ for $j \geq 1$ has a period of length n if and only if

1. $\gcd(c, n) = 1$

2. a-1 is a multiple of p for every prime p that divides n; a-1 is also a multiple of 4 if n is a multiple of 4.

LEMMA 4.2.2 Disjoint Cycles of Equal Lengths

Let n be a positive integer such that $n = k^i m$ for some integers $k > 1$ and $i \geq 2$, and m. Let c be an integer such that $\gcd(c, n) = 1$, and b be the product of all prime factors of n; b also has 4 as a factor if n is divisible by 4. Let $f_j(x) = (k^j b + 1)x + k^j c$. For every j, $1 \leq j \leq i$, the function f_j generates k^j vertex-disjoint cycles of length n/k^j on the set $\{0, 1, \ldots, n-1\}$. The vertex sets of these cycles are $A_{1,j} = \{0, k^j, \ldots, n-k^j\}, A_{2,j} = \{1, k^j+1, \ldots, n-k^j+1\}, \cdots, A_{k^j,j} = \{k^j-1, 2k^j-1, .., n-1\}$. Furthermore, there is an edge between x and y in the graph generated by f_j only if $|y-x|$ is divisible by k^j but not by k^{j+1}.

These lemmas were used in selecting the generators that create disjoint consecutive cycles in DCC linear congruential graphs. We refer the *cycle structure* of a generator (a graph) as the number of cycles and their relative lengths created by the generator (in the graph). By choosing generators according to these lemmas, the cycle structure of a DCC linear congruential graph $G(\{f_0, f_1, \ldots, f_{j-1}\}, k^i m)$ has $k^0 + k^1 + \cdots + k^{j-1}$ cycles as each f_l, $0 \leq l \leq j-1$ creates k^l cycles in which each cycle is of the same length, and the relative length of cycles among these generators is $1{:}k^{-1}{:}\cdots{:}k^{1-j}$. Furthermore, cycles among these generators are edge-disjoint. The best results for linear congruential graphs were obtained by this arrangement. However, the above lemmas cannot be used for multidimensional graphs because the generators in a multidimensional graph may contain more than one variable. When we started this research, we did not have any similar lemma that would help us in obtaining generators having the same properties as the generators used in DCC linear congruential graphs. We therefore started initially to derive some generators that create edge-disjoint cycles by computers. The numbers of cycles may not be consecutive among generators and the cycles of a generator may not have the same length. Our efforts are therefore to study the properties of multidimensional generators and to search for the rules that can create disjoint consecutive cycles.

We first investigated the graphs in the two-dimensional space to see how different it is from the one-dimensional case. Our subsequent results are then all based on two-dimensional graphs which have $\vec{s} = (2^i, s_2)$ [32]. Any vertex will be represented by either the vector form $\vec{x}$ or the (x, y) form where $0 \leq x \leq 2^i - 1, 0 \leq y \leq s_2 - 1$. We believe that all results for the two-dimensional case could be generalized to more dimensions.

4.3 The Cycle Structure of $G(F, (2^i, s_2))$

After constructing two-dimensional graphs for some time and observing those generators that were giving good results as far as the diameter is concerned, we could see that in many cases these good results were obtained by generators that generate disjoint consecutive cycles. Since the best results of one-dimensional linear congruential graphs were obtained by choosing generators that create disjoint consecutive cycles in the graphs, our interest is therefore to search for the rules that can ensure the same cycle structure in a two-dimensional graph. Eventually, we started to see some patterns in the constants of generators that can create 2^j cycles, $j \geq 0$ and edge-disjoint cycles. At the late stage of this research, we were able to prove some of the patterns we observed and they are given in this section. By following these properties and some empirical results, we provide a solution of constructing disjoint consecutive cycles in the two-dimensional graphs of $\vec{s} = (2^i, s_2)$. We will first give four definitions that are necessary for our discussion.

DEFINITION 4.3.1 Extension of a Graph

We define the extension of a two-dimensional graph $G(F, (2^i, s_2))$ (we will say the graph G is extended) to be the graph $G(F, (2^{i+1}, s_2))$.

DEFINITION 4.3.2 Regular Generators

Let G_2 be the extension of $G_1 = G(F, (2^i, s_2))$. We say that a generator is regular with respect to s_2 *if it creates the same cycle structure in G_1 and G_2. Similarly, a* *cycle structure is regular if the generator that creates it is regular. That is, the cycle structure of G_1 is preserved in G_2 when G_1 is extended.*

We are not interested in any generator that is not regular because its cycle structure lacks extensibility.

DEFINITION 4.3.3 Edge-Change of a Graph

Let $G(\{f\},(2^i,s_2))$ be a two-dimensional graph. We say that a vertex $\vec{x} = (x_1,y_1)$ of G has an edge-change when G is extended if $f(\vec{x}) \bmod (2^i,s_2) \neq f(\vec{x}) \bmod (2^{i+1},s_2)$.

The reason why we use the term *edge-change* is because for all edges $((x_1,y_1),(x_2,y_2))$ in G, if (x_1,y_1) is the vertex having an edge-change when G is extended, then the edge $((x_1,y_1),(x_2,y_2))$ in G is to be removed; a new edge $((x_1,y_1),(x_2+2^i,y_2))$ will be added at (x_1,y_1). Therefore, from (x_1,y_1) point of view, its edge is changed. The proof for this will follow.

DEFINITION 4.3.4 Half-Symmetry of a Graph

Let $G(\{f\},(2^i,s_2))$ be a two-dimensional graph. We say that the graph G is half-symmetric in terms of its two vertex subsets V_1 and V_2 where $V_1 = \{(x,y)|0 \leq x \leq 2^{i-1}-1, 0 \leq y \leq s_2-1\}$ and $V_2 = \{(x,y)|2^{i-1} \leq x \leq 2^i-1, 0 \leq y \leq s_2-1\}$, if the following condition is satisfied: For $(x_1,y_1) \in V_1$ there is an edge from (x_1,y_1) to (x_2,y_2) if and only if for $(x_1+2^{i-1},y_1) \in V_2$ there is an edge from (x_1+2^{i-1},y_1) to (x_2+2^{i-1},y_2).

The above four definitions can also be applied to the one-dimensional linear congruential graphs. The following lemma proves that when a two-dimensional (or one-dimensional) graph $G_1(\{f\},(2^i,s_2))$ is extended, the resulting graph G_2 is half-symmetric in terms of G_1.

LEMMA 4.3.1 Symmetry of Extension

Let $G_1(\{f\},(2^i,s_2))$ be a two-dimensional linear congruential graph and G_2 be the extension of G_1 where $f(\vec{x}) = \vec{x}A + \vec{b}$, $A = \begin{pmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{pmatrix}$, and $\vec{b} = (b_1,b_2)$. If a_{11} is odd, then for all edges $((x_1,y_1),(x_2,y_2))$ in G_1

1. *if* (x_1, y_1) *is a vertex not having an edge-change then* $((x_1 + 2^i, y_1), (x_2 + 2^i, y_2))$ *is an edge in* G_2.

2. *if* (x_1, y_1) *is a vertex having an edge-change then* $((x_1 + 2^i, y_1), (x_2, y_2))$ *is an edge in* G_2.

Proof:

Edges are constructed by applying the generator $f(\vec{x})$ successively to vertices. In G_1, an edge from (x_1, y_1) to (x_2, y_2) is calculated by the formula,

$$(x_2, y_2) = (x_1, y_1) + \begin{pmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{pmatrix} + (b_1, b_2) \bmod (2^i, s_2)$$

and it can be expressed as

$$\begin{aligned} x_2 &= (a_{11}x_1 + a_{21}y_1 + b_1) \bmod 2^i && (1) \\ y_2 &= (a_{22}y_1 + b_2) \bmod s_2. && (2) \end{aligned}$$

When G_1 is extended, i.e., in G_2,

$$\begin{aligned} x_2' &= (a_{11}x_1 + a_{21}y_1 + b_1) \bmod 2^{i+1} && (3) \\ y_2' &= (a_{22}y_1 + b_2) \bmod s_2. && (4) \end{aligned}$$

From (2) and (4),

$$y_2' = y_2. \qquad (5)$$

1. if $x_2' = x_2$ then there is an edge from (x_1, y_1) to (x_2, y_2) as it is in G_1. That is, there is no edge-change for the vertex (x_1, y_1) in G_1. We now show that there is a corresponding edge from $(x_1 + 2^i, y_1)$ to $(x_2 + 2^i, y_2)$ in G_2. Let

$$\begin{aligned} x_2'' &= (a_{11}(x_1 + 2^i) + a_{21}y_1 + b_1) \bmod 2^{i+1} && (6) \\ y_2'' &= (a_{22}y_1 + b_2) \bmod s_2. && (7) \end{aligned}$$

From (2) and (7),

$$y_2'' = y_2. \qquad (8)$$

Since $x_2' = x_2$, $a_{11}x_1 + a_{21}y_1 + b_1 = C2^i + D$ where C is even and $0 <= D < 2^i$. a_{11} is odd by definition. Therefore, (6) – (3) are reduced as follow.

$$\begin{aligned}
x_2'' - x_2' &= (C2^i + D + a_{11}2^i) \bmod 2^{i+1} - (C2^i + D) \bmod 2^{i+1} \\
&= ((C/2)2^{i+1} + D + ((a_{11}-1)/2)2^{i+1} + 2^i) \bmod 2^{i+1} \\
&\quad - ((C/2)2^{i+1} + D) \bmod 2^{i+1} \\
&= D + 2^i - D \\
&= 2^i.
\end{aligned}$$

So $x_2'' = x_2' + 2^i = x_2 + 2^i$.

Thus there is an edge from $(x_1 + 2^i, y_1)$ to $(x_2 + 2^i, y_2)$.

2. if $x_2' \neq x_2$ then $x_2' = x_2 + 2^i$. The original edge $((x_1, y_1), (x_2, y_2))$ is modified to a new one, $((x_1, y_1), (x_2 + 2^i, y_2))$. By definition, an edge-change is at (x_1, y_1). We now show that there must be an edge from $(x_1 + 2^i, y_1)$ to (x_2, y_2) in G_2. Let

$$x_2'' = (a_{11}(x_1 + 2^i) + a_{21}y_1 + b_1) \bmod 2^{i+1} \quad (9)$$

$$y_2'' = (a_{22}y_1 + b_2) \bmod s_2. \quad (10)$$

From (10) and (2),

$$y_2'' = y_2. \quad (11)$$

Since $x_2' \neq x_2$, $a_{11}x_1 + a_{21}y_1 + b_1 = C2^i + D$ where C is odd and $0 <= D < 2^i$.

$$\begin{aligned}
x_2'' - x_2' &= (C2^i + D + a_{11}2^i) \bmod 2^{i+1} - (C2^i + D) \bmod 2^{i+1} \\
&= (((C-1)/2)2^{i+1} + 2^i + D + ((a-1)/2)2^{i+1} + 2^i) \bmod 2^{i+1} \\
&\quad -(((C-1)/2)2^{i+1} + 2^i + D) \bmod 2^{i+1} \\
&= D - 2^i - D \\
&= -2^i.
\end{aligned}$$

So $x_2'' = x_2' - 2^i = x_2 + 2^i - 2^i = x_2$.

Thus there is an edge from $(x_1 + 2^i, y_1)$ to (x_2, y_2).

□

We have shown that for every edge in G_1, there is a corresponding edge in G_2 and its source vertex is located between 2^i and $2^{i+1}-1$. There are two formulas for this correspondence. Therefore, G_2 is half-symmetric. As shown in this proof, the value of y is not relevant to this half-symmetric property since $a_{12}=0$. This property is thus also true for a one-dimensional linear congruential graph. An example of this half-symmetric property is therefore given for a one-dimensional graph in Figure 3.10 on page 41. In the figure, vertices having edge-changes are those vertices incident with dashed edges. Furthermore, if $x_2' = x_2$ for all x_1, that is, there is no edge-change for all vertices in G_1, then there are two identical and unconnected copies of G_1 and one of them is shifted by 2^i at the x coordinates of all its vertices.

The following two lemmas further explain the properties of the generators used in DCC linear congruential graphs. They will be used in the proof of the cycle structure in the two-dimensional case.

LEMMA 4.3.2 Even Number of Cycles in $G(\{f\}, 2^i)$

Let $f(x) = ax + c, c \neq 0$ be a generator of a one-dimensional graph $G(F, 2^i)$ and the leading constant a is as in Lemma 4.2.1 or Lemma 4.2.2. $f(x)$ will create an even number of cycles in G if c is even.

Proof:

Since a is as in Lemma 4.2.1 or Lemma 4.2.2, $a = 2^j b + 1$. Assume c is even, let $c = 2^j d$, where $j \geq 1$ and $d \in N - \{0\}$. So $ax + c = (2^j b + 1)x + 2^j d$.

1. if d is odd, then $\gcd(2^i, d) = 1$. Thus $ax + c \bmod 2^i$ creates 2^j cycles by Lemma 4.2.2.

2. if d is even, let $d = 2e$ where $e \in N - \{0\}$

   - For all even vertices $x = 2y, y \in N$,

$$\begin{aligned}(2^j b + 1)(2y) + 2^j 2e \bmod 2^i &= 2^{j+1} by + 2y + 2^{j+1} e \bmod 2^i \\ &= 2((2^j + 1)y + 2^j e) \bmod 2^i \qquad (1)\end{aligned}$$

- For all odd vertices $x = 2y + 1, y \in N$,

$$(2^j b + 1)(2y + 1) + 2^j 2e \bmod 2^i$$

$$= 2^{j+1} by + 2^j b + 2y + 2^{j+1} e + 1 \bmod 2^i$$

$$= 2((2^j + 1)y + 2^j(b/2 + e)) + 1 \bmod 2^i \qquad (2)$$

(a) if e is odd and $b/2$ is even then both expressions (1) and (2) create even numbers of cycles by Lemma 4.2.2. Furthermore, expression (1) creates an even number of cycles on a subgraph of G and its vertex set is formed by all even vertices and has a size 2^{i-1}. Expression (2) creates an even number of cycles on a subgraph of G and its vertex set is formed by all odd vertices of the same size.

(b) if e is odd and $b/2$ is odd then expression (1) creates an even number of cycles by Lemma 4.2.2. Expression (2) is a linear function that is not in Lemma 4.2.2. We can further divide those odd vertices into two sets and apply this technique recursively. In the worst case, we may reach a graph of a single vertex, i.e., a self-loop cycle. Since we have a graph of size 2^i and we divide the vertex set into two subsets, the number of cycles therefore is even.

(c) if e is even and $b/2$ is odd, the same reasoning as above can be used.

(d) if e is even and $b/2$ is even, both expressions (1) and (2) are not in Lemma 4.2.2, we have to apply both of them by this technique recursively. In the worst case, we will have 2^i self-loop cycles, which is still an even number.

$\square$

LEMMA 4.3.3 Number of Edge-Changes in $G(\{f\}, 2^i)$

Let $G(\{f\}, 2^i)$ be a one-dimensional graph and G_1 be the extension of G where $f(x) = ax + c, c \neq 0$ and the leading constant a is as in Lemma 4.2.1 or Lemma 4.2.2. When G is extended, the number of edge-changes in G is odd if c is odd, and is even if c is even.

Proof:

This is proved by contradiction. Let G_1 be the extension of G.

1. c is odd: $ax + c$ is as in Lemma 4.2.1 and the cycle it generates must be Hamiltonian in G_1. By Lemma 4.3.1, we know when the graph G is extended and there is no edge-change, there are two unconnected copies of G. This is a contradiction to Lemma 4.2.1. If the number of edge-changes in G is other even numbers, edges will travel between these two copies of G. Let (x_1, y_1) be a vertex having an edge-change. There must be an edge $((x_1+2^t, y_1), (x_2, y_2))$. We count them as a pair and there are an even number of them. Starting from a vertex and traveling between these two copies of G will use only one edge in the above pair and will be back to the same cycle as we started with, since there are even number of pairs. Furthermore, we cannot go to the other cycle by the unused edge in any of the pairs above since they are all skipped. Therefore, there is no Hamiltonian cycle and this contradicts to Lemma 4.2.1. So the number of edge-changes must be odd.

2. c is even: Clearly, two cycles can be connected to be a new one by one edge-change. Two cycles remain disconnected when the edge-change is even as shown above. By Lemma 4.3.2, $ax + c$ creates an even number of cycles in G_1. If the number of edge-changes is odd, it has different parity to the number of cycles. There must be one cycle left in G with only one edge-change. Therefore, it will be connected to the corresponding cycle to form a new one. The total number of cycles therefore is reduced by 1 and become an odd number of cycles. This is a contradiction to Lemma 4.3.2. So the number of edge-changes must be even.

□

We are now stating three lemmas and some empirical results for the cycle structure of G(F,$(2^t, s_2)$). They are the lemmas for constructing a regular cycle structure, Hamiltonian cycles, edge-disjoint cycles, and an algorithm for identifying the cycle structure of a generator.

LEMMA 4.3.4 Regular Cycle Structures in $G(\{f\},(2^i,2k+1))$

Let $G_1(\{f\},(2^i,2k+1))$ be a two-dimensional linear congruential graph and G_2 be the extension of G_1 where k is an even integer and $f(\vec{x}) = \vec{x}A + \vec{b}$. Let $A = \begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix}$, $\vec{b} = (b_1, b_2)$, and a_{11}, b_1 satisfy Lemma 4.2.1. If $\gcd(b_2, 2k+1) = 1$ then $f(\vec{x})$ will generate the same cycle structure in both G_2 and G_1.

Actually, the regular cycle structure created by f in Lemma 4.3.4 is a Hamiltonian cycle. That is, when G_1 is extended, it is a Hamiltonian cycle being preserved in G_2. To be more specific, we give the following lemma, which proves that a Hamiltonian cycle is generated by f in a graph $G(F,(2^i,2k+1))$ for $i \geq 0$.

LEMMA 4.3.5 Hamiltonian Cycles in a Two-dimensional Graph

Let $f(\vec{x}) = \vec{x}A + \vec{b}$ be a generator of a two-dimensional linear congruential graph $G(F,(2^i,2k+1))$ where k is even, $A = \begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix}$, and $\vec{b} = (b_1, b_2)$. If $a_{11}x + b_1$ generates a Hamiltonian cycle in $\{0,1,\ldots,2^i-1\}$ and $\gcd(b_2,2k+1) = 1$ then $f(\vec{x})$ also generates a Hamiltonian cycle in G.

Proof:

This proof is done by induction on the variable i. Let $\vec{x} = (x, y)$.

1. $i = 0$: Since x has only one value, zero, the x coordinate of any destination vertex is the same as that of the source for all vertices. This is illustrated below.

   $y = 0$: $x = (a_{11}(0) + a_{21}(0) + b_1) \bmod 2^0 = 0$

   $y = 1$: $x = (a_{11}(0) + a_{21}(1) + b_1) \bmod 2^0 = 0$

   $y = 2$: $x = (a_{11}(0) + a_{21}(2) + b_1) \bmod 2^0 = 0$

   $\vdots$

   $y = 2k$: $x = (a_{11}(0) + a_{21}(2k) + b_1) \bmod 2^0 = 0$

   From x's point of view, they are all self-loop vertices. Since $a_{22} = 1$ and $\gcd(b_2, 2k+1) = 1$, the y sequence is a Hamiltonian cycle in $\{0,1,\ldots,2k\}$. No matter how the y sequence varies, it will connect all these self-loop vertices. Therefore, $G(\{f\},(0,2k+1))$ is a Hamiltonian cycle.

2. $i=n$: Assume $G(\{f\},(2^n,2k+1))$ is a Hamiltonian cycle. We group vertices into $2k+1$ sets in which each set has the same y value:

   $set_0 = \{(x,y)|0 \leq x \leq 2^n - 1 \text{ and } y = 0\}$

   $set_1 = \{(x,y)|0 \leq x \leq 2^n - 1 \text{ and } y = 1\}$

   $set_2 = \{(x,y)|0 \leq x \leq 2^n - 1 \text{ and } y = 2\}$

   $\vdots$

   $set_{2k} = \{(x,y)|0 \leq x \leq 2^n - 1 \text{ and } y = 2k\}$

   Since $G(\{f\},(2^n,2k+1))$ is a Hamiltonian cycle by assumption, each above set not only has 2^n vertices but also they are all connected in the same cycle. We are therefore able to prove that two copies of $G(\{f\},(2^n,2k+1))$ (two Hamiltonian cycles) are connected into one cycle by using the total number of edge-changes in G.

3. $i=n+1$: When the graph $G(\{f\},(2^n,2k+1))$ is extended, we know that the cycle structure can be preserved if the number of edge-changes is odd by Lemma 4.3.3.

   (a) a_{21} is even: since $(a_{11}x+b_1) \bmod 2^i$ is a Hamiltonian cycle, b_1 is odd.

   $y=0: (a_{11}x+0+b_1) \bmod 2^{n+1} = (a_{11}x+odd_0) \bmod 2^{n+1}$, which implies an odd number of edge-changes in set_0

   $y=1: (a_{11}x+even(1)+b_1) \bmod 2^{n+1} = (a_{11}x+odd_1) \bmod 2^{n+1}$, which implies an odd number of edge-changes in set_1

   $\vdots$

   $y=2k: (a_{11}x+even(2k)+b_1) \bmod 2^{n+1} = (a_{11}x+odd_{2k}) \bmod 2^{n+1}$, which implies an odd number of edge-changes in set_{2k}

   By Lemma 4.3.3, each of the above functions has odd number of edge-changes when G is extended. No matter how the y sequence varies, there are a total of $(2k + 1) * odd$ number of edge-changes, which is an odd number. Therefore, two Hamiltonian cycles will be connected to form a new Hamiltonian cycle when G is extended.

(b) a_{21} is odd:

$y = 0 : (a_{11}x + 0 + b_1) \bmod 2^{n+1} = (a_{11}x + odd_0) \bmod 2^{n+1}$, which implies an odd number of edge-changes in set_0

$y = 1 : (a_{11}x + odd(1) + b_1) \bmod 2^{n+1} = (a_{11}x + even_1) \bmod 2^{n+1}$, which implies an even number of edge-changes in set_1

$\vdots$

$y = 2k : (a_{11}x + odd(2k) + b_1) \bmod 2^{i+1} = (a_{11}x + odd_{2k}) \bmod 2^{n+1}$, which implies an odd number of edge-changes in set_{2k}

There are k even-numbers of cycles by Lemma 4.3.2 and $k+1$ Hamiltonian cycles by Lemma 4.2.1. By Lemma 4.3.3, for those even-numbers of cycles, there are even-numbers of edge-changes and odd-numbers of edge-changes for Hamiltonian cycles. There are therefore odd-number of edge-changes in G. So two Hamiltonian cycles are connectrd to form a new Hamiltonian cycle when G is extended.

□

Observation: Regular Cycle Structures of $G(F, (2^i, s_2))$

The above lemma gives only sufficient conditions on preserving the cycle structure and these conditions are not necessary. There are more cases that preserve the cycle structure of a graph. We have the following observations:

1. $f(\vec{x})$ satisfies the condition in Lemma 4.3.4, but k is odd. In this case, the cycle structure (a Hamiltonian cycle) is preserved when a_{21} is even.

2. b_1 is even and the rest of the conditions of Lemma 4.3.4 are retained. In this case, let $b_1 = 2l$ where $l \in N - \{0\}$. Cycle structures are preserved when

   (a) If l is odd then $a_{21} \neq 2^j - (k \bmod 2^j) + 2^j c$

(b) If l is even then $a_{21} \neq (k \bmod 2^j) + 2^j c$

where 2^j is derived from a_{11}, which has the form $(2 * 2^j + 1)$ in Lemma 4.2.2 and c $\in$ N.

3. $gcd(a_{22}, s_2) = 1$, $\gcd(b_2, s_2) \neq 1$, and the rest of the conditions of Lemma 4.3.4 are retained. In this case, cycle structures are preserved in many cases.

We have the following algorithm that can be used to identify the cycle structure of two-dimensional generators. It therefore serves the same purpose as Lemma 4.2.1 and Lemma 4.2.2 in the one-dimensional case. The restrictions for using this algorithm are the same as for Lemma 4.3.4 but b_1 can be any number and s_2 can be any odd number.

Algorithm 4.3.1 Identifying the Cycle Structure in $G(\{f\}, (2^l, s_2))$

1. list the s_2 linear functions in the generator f by replacing variable y with its actual value:

(a) $y = 0$: $a_{11}x + a_{21}(0) + b_1$

(b) $y = 1$: $a_{11}x + a_{21}(1) + b_1$

(c) $y = 2$: $a_{11}x + a_{21}(2) + b_1$

$\vdots$

(d) $y = s_2 - 1$: $a_{11}x + a_{21}(s_2 - 1) + b_1$

2. Sum up the terms $a_{21}y + b_1$ to get s_2 linear functions in which each one is in the one-dimensional form:

(a) $y = 0$: $a_{11}x + (0) + b_1 = a_{11}x + sum_0$

(b) $y = 1$: $a_{11}x + a_{21}(1) + b_1 = a_{11}x + sum_1$

(c) $y = 2$: $a_{11}x + a_{21}(2) + b_1 = a_{11}x + sum_2$

$\vdots$

(d) $y = s_2 - 1$: $a_{11}x + a_{21}(s_2 - 1) + b_1 = a_{11}x + sum_{s_2-1}$

3. *Based on Lemma 4.2.1 and Lemma 4.2.2, classify the above linear functions into pairs according to the following rules:*

   (a) *linear functions which have the same cycle structure are grouped into a pair first. For example,*

      i. *5x+1 and 5x+3: both generate Hamiltonian cycles in* $\{0,\ldots,2^i-1\}$.

      ii. *9x+2 and 9x+6: both generate 2 cycles in* $\{0,\ldots,2^i-1\}$.

      iii. *17x+4 and 17x+12: both generate 4 cycles in* $\{0,\ldots,2^i-1\}$.

      iv. *5x+4 and 5x+8: both generate irregular cycles in* $\{0,\ldots,2^i-1\}$.

   (b) *two linear functions with even* b_2 *are grouped into a pair if the above rule has not finished this pairing. For example, 5x+2 and 5x+4 are in a pair.*

4. *Since* s_2 *is odd, there is a linear function that is not in any pair. The cycle structure(regular or irregular) generated by* f *in* G *is exactly the same as the cycle structure generated by this unpaired linear function in* $\{0,1,\ldots,2^i-1\}$.

Since we use 2^i in the first dimension of our graphs, a_{11} in f should be chosen to satisfy Lemma 4.2.1 or Lemma 4.2.2. Therefore this unpaired linear function and hence the two-dimensional generator f will create either 2^j cycles, $0 \le j \le i$, or an irregular cycle structure in its corresponding graphs. Thus, each cycle created by this two-dimensional generator f is s_2 times longer than the corresponding cycle created by the unpaired linear function in $\{0,1,\ldots,2^i-1\}$. This algorithm gives us a convenient way to identify the cycle structure of a two-dimensional generator. Therefore, we can use it to derive generators that generate desirable cycle structures. Using this algorithm to identify cycle structures should be faster than using computers if s_2 is small.

LEMMA 4.3.6 Edge-Disjoint Cycles in Graphs of Degree 4

Let $G(\{f_1,f_2\},(2^i,s_2))$ *be a two-dimensional linear congruential graph and* $f_1(\vec{x}) = \vec{x}A+\vec{b}$, $f_2(\vec{x}) = \vec{x}C+\vec{d}$ *where* $A = \begin{pmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{pmatrix}$, $\vec{b} = (b_1,b_2)$, $C = \begin{pmatrix} c_{11} & 0 \\ c_{21} & c_{22} \end{pmatrix}$, *and*

$\vec{d} = (d_1, d_2)$. *Generators* $f_1(\vec{x})$ *and* $f_2(\vec{x})$ *will create edge-disjoint cycles in* G *if* a_{11}, c_{11} *are as in Lemma 4.2.1 or Lemma 4.2.2,* b_1, d_1 *are of different parities, and* a_{21}, c_{21} *are of the same parity.*

Proof:

Consider any vertex (x, y) in G, edges are constructed by f_1 and f_2.

- in f_1, $x' = (a_{11}x + a_{21}y + b_1) \bmod 2^i$
- in f_2, $x' = (c_{11}x + c_{21}y + d_1) \bmod 2^i$

Since b_1, d_1 have different parities and a_{21},c_{21} have the same parity, $a_{21}y + b_1$ and $c_{21}y + d_1$ have different parities when y is the same and this is true for all y. Therefore, if one of them is odd, the other is even for all y.

1. for the odd one, since a_{11} and c_{11} are as in Lemma 4.2.1 or Lemma 4.2.2, the x sequence, $x' = f(x) \bmod 2^i$, is a sequence such that x' and x have different parities by Lemma 4.2.1.
2. for the even one, since a_{11} and c_{11} are as in Lemma 4.2.1 or Lemma 4.2.2, the x sequence, $x' = f(x) \bmod 2^i$, is a sequence such that x' and x have the same parity by Lemma 4.3.2.

Therefore, from any (x, y), f_1 and f_2 will lead the vertex to vertices whose x components have different parities. The destination vertices generated by f_1 and f_2 cannot be the same for all y. They are therefore edge-disjoint.

□

4.4 Properties of Two-dimensional Linear Congruential Graphs

We will study some graph-theoretic properties of two-dimensional linear congruential graphs in this section.

Size: The sizes of a hypercube and a de Bruijn graph can be calculated from their input parameters. An upper bound exists for the size of a Cayley graph and this bound is decided by the number of symbols used in the permutations. Unlike these graphs, the size of a two-dimensional linear congruential graph is determined by the vector $\vec{s}$. We are able to construct graphs of any given size without any upper bound. An odd-size graph can also be constructed for any even degree. However, if we want to obtain graphs of low diameter we should use the size equal to $(a^i * b)$ where a, b are relative primes.

Diameter: Searching for generators that can produce graphs of small diameters was one of the primary goals of this thesis. We have experimentally searched for large regular graphs of small diameters. The degrees of these graphs range from 3 to 10 and their sizes range from 576 to 147456. The results of this experiment are summarized in Tables 4.1, 4.2, and 4.3. The upper right corner of a table entry shows the diameter of the corresponding degree and size, the lower part of the entry cotains the generators that achieve this diameter. In Table 4.1 and Table 4.2, all graphs are 12.5% larger in size than the DCC linear congruential graphs of the same degree and diameter. In Table 4.3, graphs are 25.5% larger. A detailed comparison on the sizes of graphs is given in Chapter 5. For graphs with up to 20480 vertices, we have calculated by computer the distances between all pairs of vertices to obtain the diameters of these graphs. It is a very time-consuming process. For graphs with more than 20480 vertices, we therefore only check segments of vertices. We believe it is sufficient to do so because a two-dimensional linear congruential graph of order (2^{i+1},c) is half-symmetric from the view of the (2^i,c) graph since it can be decomposed into two copies of (2^i,c) graphs by Lemma 4.3.1. If the two-dimensional graph of order (2^k,c) is symmetric and we choose vertices correctly, then the number of vertices that must be checked in the graph of order (2^i,c) is 2^{i-k}.

It is possible to construct a graph having different diameters for the same degree and size. That is, a graph's diameter varies with its chosen generators. The

Size / Deg	9216	18432	36864	73728	147456
	1024 * 9	2048 * 9	4096 * 9	8192 * 9	16384 * 9
3				5 0 0 1 3 4 $\boxed{19}$ 9 0 0 2 2 2	5 0 0 1 3 4 $\boxed{20}$ 9 0 0 2 2 1
4	5 0 0 1 3 5 $\boxed{10}$ 9 0 0 4 2 2	5 0 0 1 3 5 $\boxed{11}$ 9 0 0 4 2 2	5 0 0 1 2 1 $\boxed{12}$ 9 0 0 5 1 1		5 0 0 1 3 4 $\boxed{13}$ 9 0 0 2 2 1
5	5 0 0 1 6 1 $\boxed{8}$ 9 0 0 1 5 2 17 0 0 1 6 4	5 0 0 1 7 1 $\boxed{9}$ 9 0 0 1 7 2 17 0 0 1 4 3		5 0 0 1 7 1 $\boxed{10}$ 9 0 0 1 7 2 17 0 0 1 4 3	5 0 0 1 7 1 $\boxed{11}$ 9 0 0 1 7 2 17 0 0 1 4 3
6	5 0 0 1 7 1 $\boxed{7}$ 9 0 0 1 7 2 17 0 0 1 4 3		5 0 0 1 9 1 $\boxed{8}$ 9 0 0 1 8 2 17 0 0 1 9 3		5 0 0 1 7 1 $\boxed{9}$ 9 0 0 1 7 2 17 0 0 1 4 3

Table 4.1: Graphs of $\vec{s} = (2^i, 9)$

problem of determining the diameter of a two-dimensional linear congruential graph is open. Actually, even in the one-dimensional case, the diameter of a DCC linear congruential graph is not known exactly. This should not be surprising as the diameters of chordal rings are not exactly known [4, 17] and chordal rings constitute the simplest case of linear congruential graphs. For this reason, diameters of all graphs had to be calculated by computer.

Regularity: We have chosen the generators such that each generator constructs edge-disjoint cycles in our graphs. That is, vertices are partitioned into disjoint sets by the cycles. By definition, each generator adds two edges to a vertex, so it is regular for even-degree graphs. We have also chosen the matching generator to generate an edge for every two vertices, so it is regular for odd-degree graphs of even graph size.

Size	576	1152	2304	4608	9216	18432	36864	73728	147456
Deg	64 • 9	128 • 9	256 • 9	512 • 9	1024 • 9	2048 • 9	4096 • 9	8192 • 9	16384 • 9
7		5 0 0 1 5 1 $\boxed{5}$ 9 0 0 1 6 2 17 0 0 1 2 3 33 0 0 1 2 8		5 0 0 1 5 1 $\boxed{6}$ 9 0 0 1 6 2 17 0 0 1 2 3 33 0 0 1 2 8		5 0 0 1 5 1 $\boxed{7}$ 9 0 0 1 6 2 17 0 0 1 2 3 33 0 0 1 2 8			
8	5 0 0 1 5 1 $\boxed{4}$ 9 0 0 1 6 2 17 0 0 1 2 3 33 0 0 1 2 8		5 0 0 1 5 1 $\boxed{5}$ 9 0 0 1 6 2 17 0 0 1 2 3 33 0 0 1 3 7		5 0 0 1 5 1 $\boxed{6}$ 9 0 0 1 6 2 17 0 0 1 2 3 33 0 0 1 3 7				5 0 0 1 15 1 $\boxed{8}$ 9 0 0 1 14 2 17 0 0 1 16 3 33 0 0 1 4 7
9			5 0 0 1 2 1 $\boxed{5}$ 9 0 0 1 5 2 17 0 0 1 1 3 33 0 0 1 1 7 65 0 0 1 3 4			5 0 0 1 2 1 $\boxed{6}$ 9 0 0 1 5 2 17 0 0 1 1 3 33 0 0 1 1 7 65 0 0 1 3 4			
10		5 0 0 1 2 1 $\boxed{4}$ 9 0 0 1 5 2 17 0 0 1 1 3 33 0 0 1 1 7 65 0 0 1 3 4		5 0 0 1 2 1 $\boxed{5}$ 9 0 0 1 5 2 17 0 0 1 3 3 33 0 0 1 1 7 65 0 0 1 1 1			5 0 0 1 2 1 $\boxed{6}$ 9 0 0 1 5 2 17 0 0 1 3 3 33 0 0 1 1 7 65 0 0 1 1 1		5 0 0 1 2 1 $\boxed{7}$ 9 0 0 1 5 2 17 0 0 1 3 3 33 0 0 1 1 7 65 0 0 1 1 1

Table 4.2: Graphs of $\vec{s} = (2^i, 9)$ (continued)

Connectivity: The connectivity of a two-dimensional linear congruential graph is determined by its cycle structure, which is constructed by the chosen set of generators. To generalize the idea of connectivity in a DCC linear congruential graph, an even-degree graph $G(F, \vec{s})$ where $F = \{f_i(\vec{x}) | 1 \leq i \leq k \text{ for some } k\}$ and $\vec{s} = (2^i, c)$, has the maximum connectivity $2k$ if the set of generators has the following properties:

Property 4.4.1 Connectivity of a Two-dimensional Linear Congruential Graph

1. $f_i(\vec{x}) \bmod \vec{s} <> f_i^{-1}(\vec{x}) \bmod \vec{s}$ *for all* i
2. *The function can be arranged in a way that* $f_{i+1}(\vec{x})$ *partitions every cycle generated by* $f_i(\vec{x})$ *into two disjoint sets, any two consecutive vertices* $\vec{x}$ *and*

Deg \ Size	640	1280	2560	5120	10240	20480	40960
	128 * 5	256 * 5	512 * 5	1024 * 5	2048 * 5	4096 * 5	8192 * 5
3	5 0 $\boxed{10}$ 6 1 13 1 9 0 6 1 14 1	5 0 $\boxed{12}$ 0 1 6 8 9 0 0 1 3 4	5 0 $\boxed{13}$ 0 1 6 8 9 0 0 1 3 4	5 0 $\boxed{14}$ 10 1 5 1 9 0 10 1 6 1		5 0 $\boxed{17}$ 4 1 1 1 9 0 0 1 4 1	5 0 $\boxed{18}$ 0 1 7 1 9 0 0 1 12 1
4	5 0 $\boxed{7}$ 1 1 1 1 9 0 5 1 8 1	5 0 $\boxed{8}$ 1 1 1 1 9 0 5 1 4 1	5 0 $\boxed{9}$ 0 1 1 1 9 0 0 1 4 1			5 0 $\boxed{11}$ 1 1 1 1 9 0 3 1 4 1	5 0 $\boxed{12}$ 1 1 1 1 9 0 3 1 4 1
5	5 0 $\boxed{6}$ 4 1 5 1 9 0 4 1 2 1 17 0 0 1 4 1		5 0 $\boxed{7}$ 0 1 3 1 9 0 0 1 4 2 17 0 0 1 1 1				
6	5 0 $\boxed{5}$ 4 1 5 1 9 0 4 1 2 1 17 0 0 1 4 1	5 0 $\boxed{6}$ 0 1 3 1 9 0 0 1 2 1 17 0 0 1 1 1					

Table 4.3: Graphs of $\vec{s} = (2^i, 5)$

$f_i(\vec{x}) \bmod \vec{s}$ *on the original cycle are not in the same set, and* $f_{i+1}(\vec{x}) \bmod \vec{s} = f_i(f_i(\vec{x}) \bmod \vec{s}) \bmod \vec{s}$.

We give a proof for graphs of degree 4 and 6. The same reasoning can be used to prove graphs of higher degrees.

Proof:

1. degree 4: f_2 partitions f_1 into two disjoint cycles C_1, C_2, and $\vec{x}$ and $f_i(\vec{x}) \bmod \vec{s}$ are not on the same cycle, and $f_i(\vec{x}) \bmod \vec{s} <> f_i^{-1}(\vec{x}) \bmod \vec{s}$. We can therefore depict f_1 and f_2 as follows and demonstrate 4 vertex-disjoint paths among any pair of vertices $\vec{x_1}$ and $\vec{x_2}$. Edges of f_1 that are not adjacent to $\vec{x_1}$ or $\vec{x_2}$ are not drawn.

(a) Case 1. If both $\vec{x_1}$ and $\vec{x_2}$ are on the same cycle as shown in Figure 4.3, the paths are:

   i. path 1: $\vec{x_1}, f_1(\vec{x_1})$, part of C_2 not containing $f_1^{-1}(\vec{x_1})$ to $f_1^{-1}(\vec{x_2})$, $\vec{x_2}$

   ii. path 2: $\vec{x_1}, f_1^{-1}(\vec{x_1})$, part of C_2 not containing $f_1(\vec{x_1})$ to $f_1(\vec{x_2})$, $\vec{x_2}$

   iii. path 3: $\vec{x_1}$, one part of C_1 to $\vec{x_2}$

   iv. path 4: $\vec{x_1}$, the other part of C_1 to $\vec{x_2}$

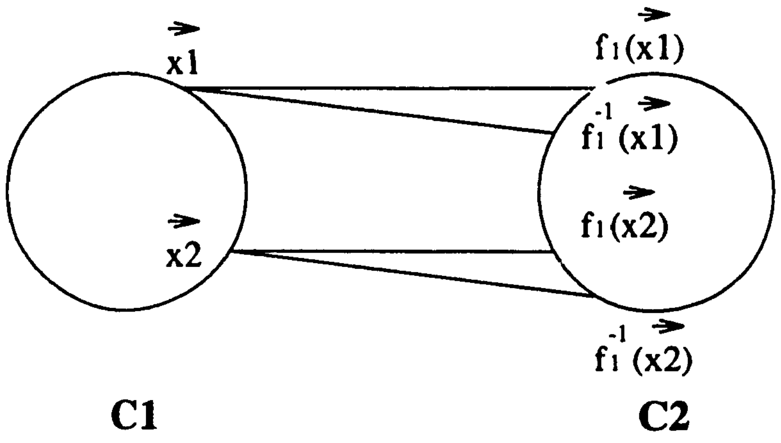


Figure 4.3: Connectivity of degree 4, $\vec{x_1}$ and $\vec{x_2}$ on the same cycle

(b) Case 2. If $\vec{x_1}$ and $\vec{x_2}$ are on different cycles as shown in Figure 4.4, the paths are:

   i. path 1: $\vec{x_1}$, $f_1(\vec{x_1})$, one part of C_2 to $\vec{x_2}$

   ii. path 2: $\vec{x_1}$, $f_1^{-1}(\vec{x_1})$, the other part of C_2 to $\vec{x_2}$

   iii. path 3: $\vec{x_1}$, part of C_1 not containing $f_1^{-1}(\vec{x_2})$ to $f_1(\vec{x_2}), \vec{x_2}$

   iv. path 4: $\vec{x_1}$, part of C_1 not containing $f_1(\vec{x_2})$ to $f_1^{-1}(\vec{x_2}), \vec{x_2}$

2. degree 6: Similar to 1, f_2 partitions f_1 into C_1 and C_2. C_1 is further refined into C_{11} and C_{12}, C_2 into C_{21} and C_{22} by f_3. Each vertex has two edges in its own cycle, two edges to the other cycle under f_3, and two edges to the other cycle under f_2 and furthermore each one is connected to

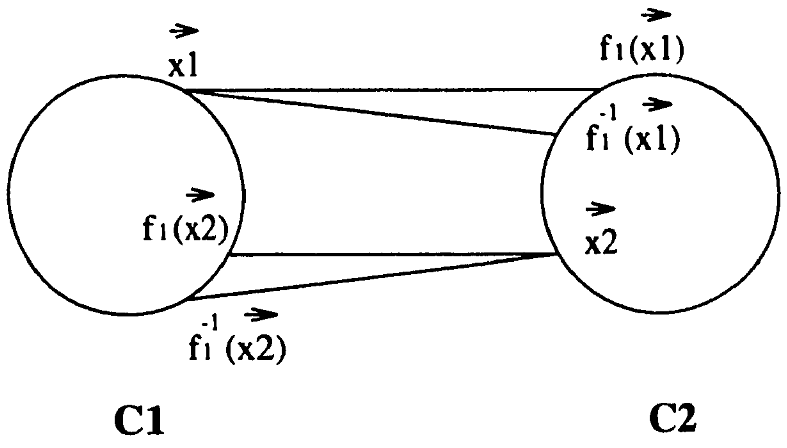


Figure 4.4: Connectivity of degree 4, $\vec{x_1}$ and $\vec{x_2}$ on different cycles

different cycles under f_3. We therefore can draw f_1, f_2, and f_3 as follows and demonstrate 6 vertex-disjoint paths.

(a) Case 1. If $\vec{x_1}$ and $\vec{x_2}$ are on the same cycle under f_2 as shown in Figure 4.5, the paths are:

  i. path 1: $\vec{x_1}$, $f_2(\vec{x_1})$, part of C_{12} to $\vec{x_2}$
  ii. path 2: $\vec{x_1}$, $f_2^{-1}(\vec{x_1})$, part of C_{12} to $\vec{x_2}$
  iii. path 3: $\vec{x_1}$, part of C_{11} to $f_2(\vec{x_2})$,$\vec{x_2}$
  iv. path 4: $\vec{x_1}$, part of C_{11} to $f_2^{-1}(\vec{x_2})$,$\vec{x_2}$
  v. path 5: $\vec{x_1}$, $f_1(\vec{x_1})$, part of C_{21} to $f_1(\vec{x_2})$, $\vec{x_2}$
  vi. path 6: $\vec{x_1}$, $f_1^{-1}(\vec{x_1})$, part of C_{22} to $f_1^{-1}(\vec{x_2})$, $\vec{x_2}$

(b) Case 2. If $\vec{x_1}$ and $\vec{x_2}$ are on different cycles under f_2 as shown in Figure 4.6, the paths are:

  i. path 1: $\vec{x_1}$, $f_1(\vec{x_1})$, part of C_{21} to $\vec{x_2}$
  ii. path 2: $\vec{x_1}$, $f_2^{-1}(\vec{x_1})$, $f_1(f_2^{-1}(\vec{x_1}))$, part of C_{21} to $\vec{x_2}$
  iii. path 3: $\vec{x_1}$, part of C_{11} to $f_1(\vec{x_2})$, $\vec{x_2}$
  iv. path 4: $\vec{x_1}$, part of C_{11} to $f_1(f_2(\vec{x_2}))$, $f_2(\vec{x_2})$, $\vec{x_2}$
  v. path 5: $\vec{x_1}$, $f_1^{-1}(\vec{x_1})$, part of C_{22} to $f_2^{-1}(\vec{x_2})$, $\vec{x_2}$

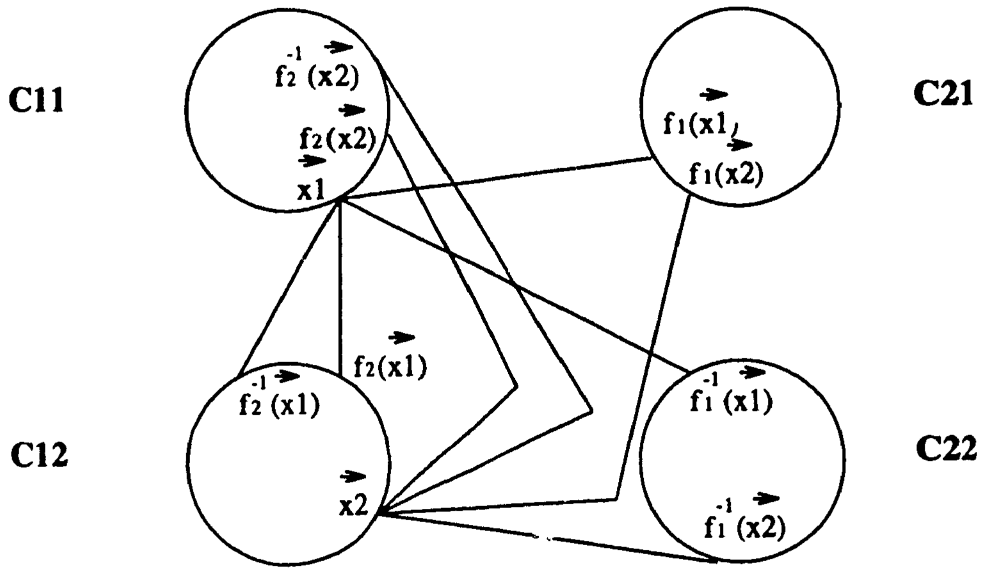


Figure 4.5: Connectivity of degree 6, $\vec{x_1}$ and $\vec{x_2}$ on the same cycle of f_2

vi. path 6: $\vec{x_1}$, $f_2(\vec{x_1})$, part of C_{12} to $f_1^{-1}(\vec{x_2})$, $\vec{x_2}$

The similar idea can be followed to prove other cases that cycles are under f_3's partition.

□

One example of a generator set that satisfies the above property is a set in which the following are true:

1. For all generators $a_{12} = a_{21} = 0$, $a_{22} = 1$, and $\gcd(b_2, s_2) = 1$.
2. There is exactly one generator whose a_{11}, b_1 are as in Lemma 4.2.1.
3. All other generators are of different numbers of cycles and their a_{11}, b_1 are as in Lemma 4.2.2.

Any generator set that satisfies the above requirements has the properties: the x and y components are independent and the y sequence is a Hamiltonian cycle itself for all the generators in the set, the cycle structure created by such a

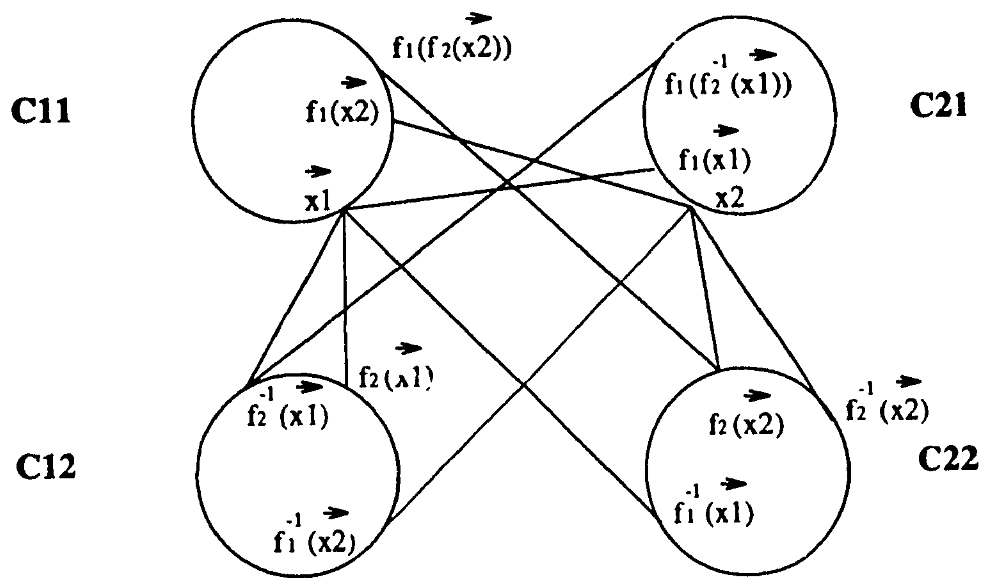


Figure 4.6: Connectivity of degree 6, $\vec{x_1}$ and $\vec{x_2}$ on different cycles of f_2

generator is thus determined by its x sequence, $x = (a_{11}x + b_1) \bmod s_1$. A two-dimensional graph with these generators therefore has exactly the same cycle structure as a one-dimensional DCC linear congruential graph because a_{11}, b_1 determine the x sequence and they are selected by Lemma 4.2.1 or 4.2.2, which are used in the DCC linear congruential graphs.

For those odd-degree graphs that their generator sets have the above property, they may not be 2k+1-connected. However they are at least 2k-connected since it contains the 2k-degree graph as a subgraph.

Extensibility: Two copies of graphs $G_1(F,(2^i,c))$ can be combined into one graph $G_2(F,(2^{i+1},c))$ with only a few edges being changed, and its cycle structure is preserved. The algorithm for the extension follows the idea of Lemma 4.3.1 and all generators of G_1 must be regular. The total number of edges remains the same after extension and only those edges that cross the 2^i boundary are modified. We give the algorithm for the extension below.

Algorithm 4.4.1 Extension of a Two-dimensional Graph

```
1. Denote the two copies of graph $G_1(F,(2^i,c))$ by $H_1, H_2$.
2. For all vertices $(x,y)$ in $H_2$
       $(x,y) = (x,y) + (2^i,0)$
3. For all vertices $(x,y)$ in $H_1$
       For all generators $f$ in $F$
           $(x_1,y_1) = f(x,y) \bmod (2^{i+1},c)$
           if $x_1 >= 2^i$ then
               delete the edges $((x,y),(x_1,y_1))$ and $((x+2^i,y),(x_1+2^i,y))$
               add the edges $((x,y),(x_1+2^i,y_1))$ and $((x+2^i,y),(x_1,y_1))$
```

4.5 Issues on Construction

In this section, we discuss several issues that can influence the construction of two-dimensional graphs. From our computer investigations of two-dimensional linear congruential graphs we observed that generators, the length of each dimension, and the matching generator used in odd-degree graphs are important parameters in searching for graphs of low diameters. We describe them in the following. In addition, we also discuss the problem of constructing graphs in a three-dimensional space.

Generator: It is now clear that the generators in both Cayley graphs and two-dimensional graphs can affect diameters. By changing a generator, we can substantially change the graph generated. Good graphs are constructed by properly choosing the parameters in generators, but it is difficult to tell the role of each constant of a generator. However, based on the matrices we used, we can classify all generators as being one of two forms (simple and complex) and explain their general effects for the two-dimensiona' case as follows.

- Simple:

  1. Matrix elements $m_{ii} \neq 0$ for all i, and $m_{ij} = 0, i \neq j$ for all i,j.

2. The generator can be rewritten to two linear functions in which each one contains exactly one variable. For example,
$$(x,y) = (x,y)\begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix} + (b_1, b_2)$$
can be expressed as $x = a_{11}x + b_1$ and $y = a_{22}y + b_2$.

3. As shown in the example above, the x, y components of any destination vertex are calculated independently by the function that contains the corresponding variable. x and y are therefore independent.

4. Row-switching is only dependent on the current row on which a vertex is located. Therefore, all vertices on one row are adjacent to the vertices in the same row.

5. The graph $G(F, (s_1, s_2))$ can be obtained by the product of the graphs, $G_1(F_1, s_1)$ and $G_2(F_2, s_2)$, which are two one-dimensional graphs and $F_i = \{a_{t,ii}x + b_{t,i} \mid 1 \leq t \leq k$, k is the number of generators in $F\}$. The product of $G_1 \circ G_2$ is defined as follows:

   Property 4.5.1 The Product of Two One-dimensional Graphs *$G_1 \circ G_2$ is a graph with vertex set $V = V(G_1) \times V(G_2)$ and the edge set $E = \{((u_1, u_2), (v_1, v_2)) \mid (u_1, v_1) \in G_1, (u_2, v_2) \in G_2\}$*

   The number of cycles in G is also the product of the number of cycles in G_1 and G_2.

6. **Property 4.5.2 Inheritance of the Cycle Structure of a One-dimensional Graph**
   The graph $G(F, (s_1, s_2))$, where $|F| = k$ and $\gcd(s_1, s_2) = 1$, inherits the cycle structure of the DCC linear congruential graph, $G_1(F_1, s_1)$ where $F_1 = \{c_i x + d_i \mid 1 \leq i \leq k\}$ if $a_{i,11} = c_i$, $b_{i,1} = d_i$, $a_{i,22} = 1$, and $\gcd(b_{i,2}, s_2) = 1$ for all generators in F.

Figure 4.7 is an example of a two-dimensional graph constructed by using generators in this form. Its diameter is 6. The Hamiltonian cycle (outer circle) is created by f_1 and the other two cycles (solid and dashed) are created by f_2.

- Complex:

  1. Matrix elements $m_{ii} \neq 0$ for all i, and $m_{ij} \neq 0, i \neq j$ for some i,j.
  2. The generator can be rewritten to two linear functions in which each one contains at least one variable. For example,
  $$(x,y) = (x,y)\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + (b_1, b_2)$$
  can be expressed as $x = a_{11}x + a_{21}y + b_1$ and $y = a_{12}x + a_{22}y + b_2$. In our empirical studies, we observed that if both a_{12} and a_{21} are non-zero and are not congruent to the size of the dimension then the generators of this type do not give regular cycle structures. We are not interested in any generator that creates an irregular cycle structure because it lacks extensibility.
  3. As shown in the above example, the x, y components of any destination vertex are calculated by a function that may contain the other component. Therefore, x and y are related.
  4. Row-switching may depend on both x's and y's values. Therefore, vertices on one row can be linked to different rows.
  5. As an edge travels across a column, a different linear function will be applied at the destination vertex. So graphs constructed by this matrix form cannot be obtained by the product of two one-dimensional graphs.
  6. The generation is more complex because vertices on different columns use different linear functions.

Figure 4.8 is an example of a two-dimensional graph constructed by using generators in this form. Its diameter is 5. The Hamiltonian cycle (outer circle) is created by f_1 and the other two cycles (solid and dashed) are created by f_2.

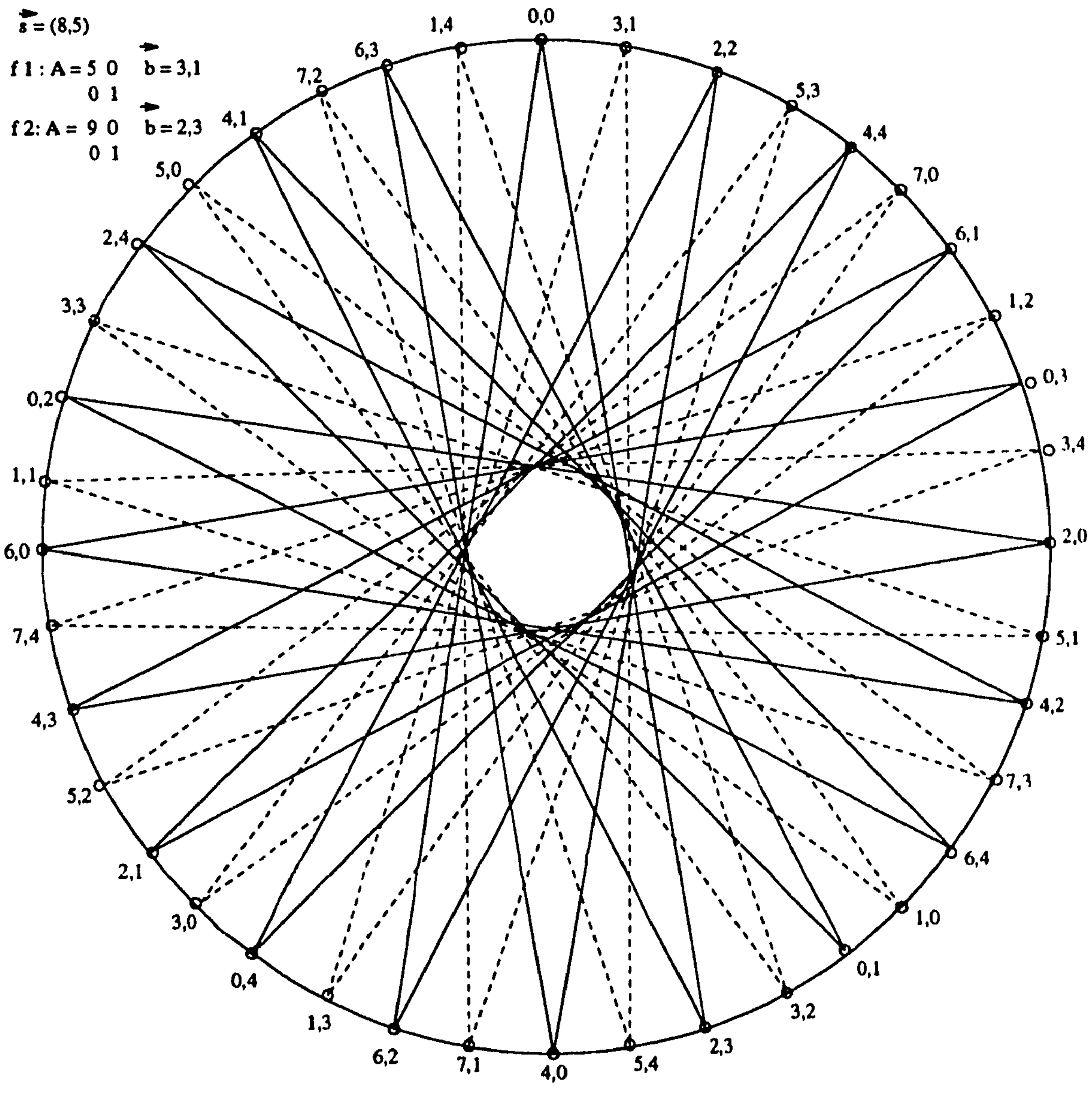


Figure 4.7: A two-dimensional graph of simple matrix form

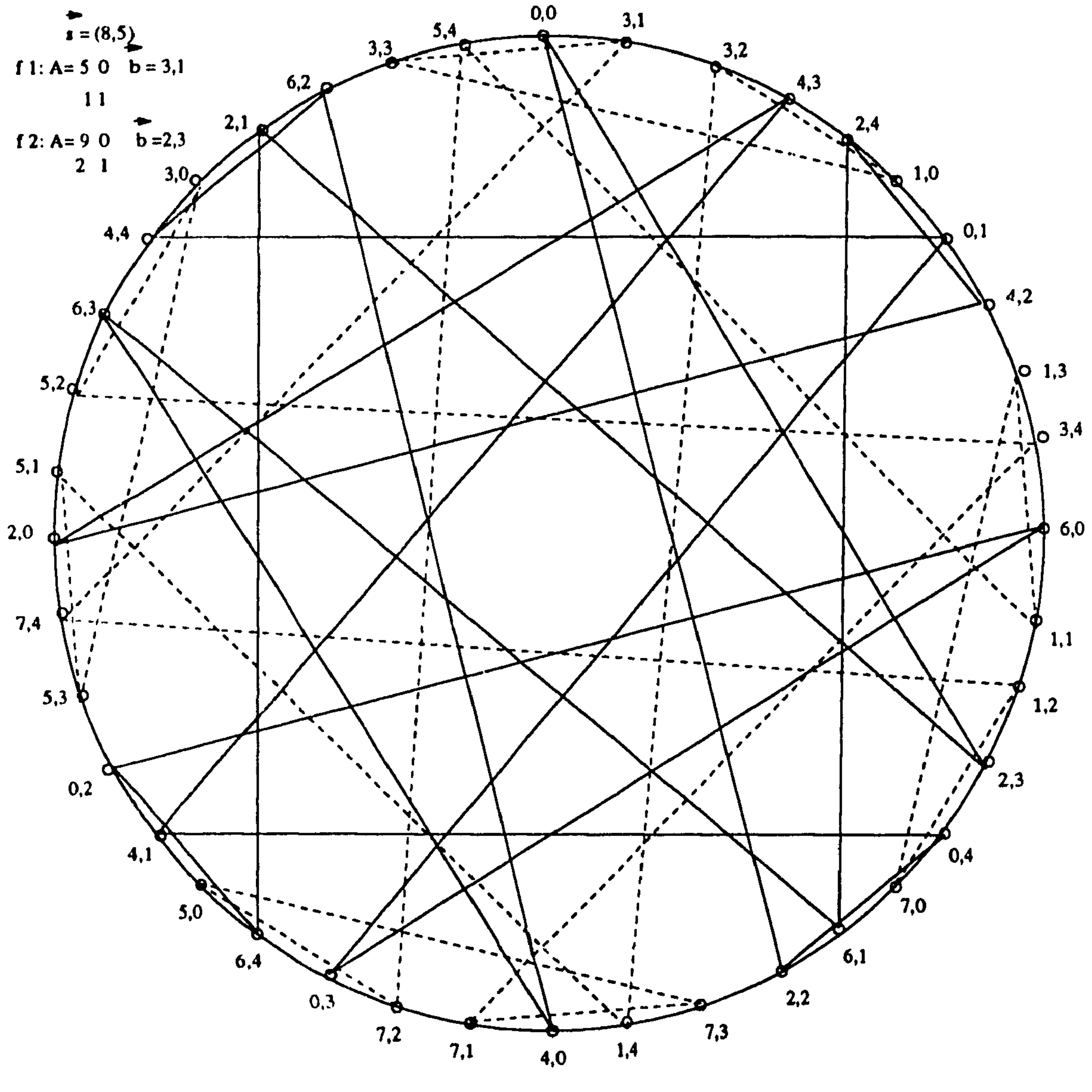


Figure 4.8: A two-dimensional graph of complex matrix form

The Chordal ring [4] has been considered as a candidate for computer networks because it has both simple structure and simple routing algorithm. Chordal rings can be constructed by the above mechanism, however, the diameters of these graphs are large. This is because their cycles are commutative, every chordal shares the same and fixed length. There is no way to reach a farther vertex by some "shortcuts". Better diameters were found by those generators that can create different cycle structures. For example, a better generator should create two cycles in which one has three-vertex jumps and the other one has thirty-vertex jumps instead of two cycles resembling each other. This can be realized by comparing the cycle structures and diameters of the graphs in Figure 4.7 and Figure 4.8.

Length of each dimension: Generators determine the edges of a two-dimensional graph. The length of each dimension compose the graph size. The best results for DCC linear congruential graphs were obtained when the size of one dimension of a graph is a large power of a prime number. We thus started our investigation by using a similar case, namely, s_1 is of the form $p^i * b$ for a prime number p. The reason why we choose 2 as p is because the function of power of 2 grows slower than powers of other prime numbers and therefore produces many more possible values that can meet the memory requirement in computers as a reasonable graph size. We can use $(2^k + 1)$ in the matrix element a_{11}, odd number in b_1 to generate a Hamiltonian cycle and $(2^k + 1)$ in a_{11}, $2^k c$ in b_1 to generate 2^k cycles in the first dimension by Lemma 4.2.1 and Lemma 4.2.2. Furthermore, by restricting $a_{22} = 1, a_{12} = a_{21} = 0$, and $\gcd(b_2, s_2) = 1$, we can construct exactly the same cycle structure in a two-dimensional graph as in a one-dimensional DCC linear congruential graph.

The question left is how to choose the length of the other dimension. To speed up the propagation of messages, any node in a computer network of degree $2k$ should be able to send and receive k messages in parallel. Many parallel algorithms are developed basing on this assumption. A graph which can be decomposed into k sets of edge-disjoint cycles can admit this parallelism. Examine the creation of cycles in a DCC linear congruential graph, it fits into this structure naturally because each generator creates a number of vertex-disjoint cycles and generators can be chosen to create edge-disjoint cycles. The graph can thus be easily decomposed according to its generators. However, 1) to ensure that these cycles are connected and 2) to guarantee a perfect matching on a vertex set, we need a Hamiltonian cycle in our graph to ease the construction. Therefore, $\gcd(s_1, s_2) = 1$. Furthermore, to preserve the cycle structure of a graph $G(F,(2^i, s_2))$ when G is extended, s_2 must follow the form of $2k+1$ where k is even. The following table contains graphs that have different lengths. They are 26.6% and 75% larger in size than the DCC linear congruential graphs of the same degree and diameter. A comparison will be given in Chapter 5.

Matching: A matching generator influences a two-dimensional graph in the following ways:

1. Like the normal generators, different matching generators can possibly create different graphs.
2. If $g(\vec{x})$ creates a perfect matching on a vertex set V when it is applied on the subset V_1, then clearly, it is also a perfect matching when applied on the subset $V - V_1$, which may also give a different result.

The following theorem enables us to simplify the task of searching for low-diameter graphs by focusing on matching generators instead of on the vertex set.

Theorem 4.5.1 $G(F, \vec{s}, g, V_1) = G(F, \vec{s}, g^{-1}, V_2)$

Let g be a matching generator of a two-dimensional linear congruential graph $G(F,(2^i,c),g,V_1)$ and $V_2 = V - V_1$. If g is of a simple matrix form then there exists a generator h such that $G(F,(2^i,c),g,V_2) = G(F,(2^i,c),h,V_1)$.

Proof:

Degree	Diameter	Size	Linear Functions	
4	8	16 * 81	$A = \begin{matrix} 5 & 0 \\ 0 & 2 \end{matrix}$	$\vec{b} =$ 1 2
		1296	$\begin{matrix} 9 & 0 \\ 0 & 4 \end{matrix}$	10 1
4	9	32 * 81	$\begin{matrix} 5 & 0 \\ 0 & 2 \end{matrix}$	1 2
		2592	$\begin{matrix} 9 & 0 \\ 0 & 4 \end{matrix}$	4 1
6	5	8 * 81	$\begin{matrix} 5 & 0 \\ 0 & 1 \end{matrix}$	2 3
		648	$\begin{matrix} 9 & 0 \\ 0 & 4 \end{matrix}$	3 2
			$\begin{matrix} 17 & 0 \\ 0 & 10 \end{matrix}$	1 3
6	6	16 * 81	$\begin{matrix} 5 & 0 \\ 0 & 1 \end{matrix}$	2 3
		1296	$\begin{matrix} 9 & 0 \\ 0 & 4 \end{matrix}$	3 2
			$\begin{matrix} 17 & 0 \\ 0 & 10 \end{matrix}$	1 3
6	6	256 * 7	$\begin{matrix} 5 & 0 \\ 1 & 1 \end{matrix}$	4 1
		1792	$\begin{matrix} 9 & 0 \\ 1 & 1 \end{matrix}$	1 1
			$\begin{matrix} 17 & 0 \\ 0 & 1 \end{matrix}$	4 2

Table 4.4: Graphs of $\vec{s} = (2^i, 81)$ and $(2^i, 7)$

Since g is a matching generator in the graph $G(F, \vec{s}, g, V_1)$, g is also a one-to-one mapping from V_2 to V_1 and $g(\vec{x})$ mod $(2^i, c) \in V_1$ for any $\vec{x} \in V_2$. Let $g(\vec{x}) = (x, y) \begin{pmatrix} a_{11} & 0 \\ 0 & a_{22} \end{pmatrix} + (b_1, b_2)$, so $x = a_{11}x + b_1$ and $y = a_{22}y + b_2$. By a theorem in [28], there exists constants c_{11}, d_1 and c_{22}, d_2 such that $c_{11}x + d_1$ is an inverse function of $a_{11}x + b_1$ and $c_{22}y + d_2$ is an inverse function of $a_{22}y + d_2$. Since the x, y values of $g(\vec{x})$ are independent, clearly, $h(\vec{x}) = (x, y) \begin{pmatrix} c_{11} & 0 \\ 0 & c_{22} \end{pmatrix} + (d_1, d_2)$ is an inverse function of $g(\vec{x})$ and the matching defined by h on V_1 is identical to the matching defined by g on V_2.

□

The above theorem concludes that we can always find a function $h(\vec{x}) = g^{-1}(\vec{x})$ on V_1 such that $G(F, \vec{s}, g, V_2) = G(F, \vec{s}, g^{-1}, V_1)$.

Three dimensions: The problems of constructing three-dimensional (or more) graphs would involve the similar issues as we did in this thesis. Namely, searching for generators that produce low-diameter graphs, investigating the cycle structures and properties of these generators, etc. We did a very detailed search for low-diameter two-dimensional graphs. The average load on a Sun workstation is approximately 60 hours/week and the estimated CPU time spent to calculate diameters of these graphs is 30000 hours. Thus we did not have enough time to search for low-diameter graphs in a three-dimensional space and this problem is still open. However, we have to point out that the difficulty of the problem increases a lot when switching from one dimension to two dimensions because we did not have knowledge of the cycle structure initially, and we feel that the problem of generating graphs in three or more dimensions will be simpler because many properties that we discovered in the two-dimensional case can be generalized without much difficulty. In particular, one can start immediately to construct edge-disjoint consecutive cycles in three-dimensional graphs by using simple matrix forms and extending the idea in Property 4.5.2.

4.6 Summary

The cycle structure of a two-dimensional generator is a new topic, which has not been investigated by anyone before our studies. Our investigation of two-dimensional linear congruential graphs indicate that a two-dimensional linear congruential graph should have a cycle structure similar to that of a DCC linear congruential graph. Namely, the cycle structure created by a set of generators should have the following properties:

1. All generators create edge-disjoint cycles.

2. All generators are regular.

3. All generators satisfy the property of consecutive cycles.

The cycle structure of a d-dimensional graph is determined by its generator set. If the generator set is of size k, the question is how to choose the $k(d^2 + d)$ constants in matrices and vectors to include the above requirements. We had found a solution for the two-dimensional case:

1. Simple matrix: In the simplest case, by restricting $a_{22} = 1$ and $\gcd(s_2, b_2) = 1$ in a generator, the cycle structure is then determined by a_{11} and b_1. If we choose a_{11} and b_1 by following Lemma 4.2.1 or Lemma 4.2.2, the resulting cycle structure of the generator is given by these lemmas.

2. Complex matrix: We have a lemma and some empirical results to create regular cycles, an algorithm to identify cycle structures based on the results in the one-dimensional case, and a sufficient condition for constructing edge-disjoint cycles. We are therefore able to use the same approach as DCC linear congruential graphs to construct disjoint consecutive cycles in two-dimensional graphs by using the complex matrix form.

This solution can be thought as a breakthrough into the multidimensional space. A good example is that operations on integers and matrices are different, however, once people established the rules of two-dimensional matrix operations, it was easy to extend them into any number of dimensions.

Multidimensional linear congruential graphs are a generalization of linear congruential graphs. Properly said, it is a very general concept for constructing graphs rather than just a single class of graphs. It provides a way to construct a variety of classes of graphs according to the choice of a generator set and a graph size. These graphs include some known families of graphs such as Chordal rings, de Bruijn graphs, etc, and include some new families. Since the properties of a multidimensional linear congruential graph are completely specified by its generator set, properties of these graphs can be analyzed by algebraic tools and proved as a whole class. A new graph therefore can inherit the known properties such as being edge-disjoint, half-symmetric, Hamiltonian, if the cycle structure of its generators is known.

We have constructed a list of graphs which have 12.5% to 75% improvement of the graph size in comparison to other constructions. If a two-dimensional linear congruential graph is Hamiltonian and its cycle structure is regular, the graph has most of the good properties in terms of network design as we have proven. In general, the advantages of two-dimensional linear congruential graphs in terms of network design can be realized by listing the disadvantages of those families of graphs discussed in Chapter 3 as below and these disadvantages are not found in two-dimensional linear congruential graphs.

- Hypercube: Its degree grows with graph size and its diameter is too large.
- Cayley graph: Its degree grows with graph size and its size is much smaller than that of a two-dimensional linear congruential graph of the same degree and diameter.
- De Bruijn graph: It is not regular in terms of the definition of an undirected simple graph. Furthermore, only even-degree graphs can be constructed and, given a graph size, only some degrees are possible.
- DCC linear graph: It is smaller in size and it is less flexible in that it is a special case of multidimensional linear congruential graphs.

In the next chapter, we will give detailed comparisons among all these graphs. These comparisons will be based on their construction methods, the size and diameter they achieve, and other graph-theoretic properties.

Chapter 5

Comparisons between Multidimensional Linear Congruential Graphs and Other Networks

5.1 Construction Methods

Table 5.1 gives a comparison of the construction methods of hypercube graphs, de Bruijn graphs, Cayley graphs, linear congruential graphs, and multidimensional linear congruential graphs. Basically, it explains how these graphs are constructed and how their graph-theoretic parameters are derived. We explain the meaning of each row in the table below,

- row 1: necessary parameters to construct the graph listed in the corresponding column
- row 2: how the graph's size is derived
- row 3: how the vertices are specified

- row 4: how the edges are derived
- row 5: how the graph's diameter is derived
- row 6: how the graph's degree is derived
- row 7: Is it a single class of graphs

Essence \ Graph	Hypercube	De Bruijn	Cayley	One-dimensional	K-dimensional
Input Parameters	n	d and D	n and a set of generators	a set of linear functions and n	a set of linear functions and $\vec{s}$
Size	2^n	d^D	$n!$ n length of a gen.	n	S1•S2• •Sk
Vertices	n-bit binary string	D-bit d-ary string	group elements	integer	a k-tuple of integers
Edges (x,y)	x and y differ in one bit	right- or left-shift of x	y=f(x)	y=f(x) mod n	y=f(x) mod (s1,s2, ,sk)
Diameters	n	D	varies with graphs	varies with graphs	varies with graphs
Degree	n	2d	(size of set)	2(size of set) or 2(size of set)+1	2(size of set) or 2(size of set)+1
Classes	Y	Y	N	N	N

Table 5.1: A comparison of construction

5.2 Graph Sizes

Table 5.2 and Table 5.3 give comparisons of the sizes of these graphs based on the same degree Δ and diameter D. Each entry of a given (Δ,D) is divided into five fields: From the top to the bottom, they are used to indicate the sizes of hypercube, Cayley, de Bruijn, DCC, and two-dimensional linear congruential graphs respectively. An empty field indicates that a graph of the corresponding (Δ,D) cannot be constructed. Therefore, we also show the sizes of some graphs having comparable (Δ,D).

Deg	Gra \ Diam	10	12	13	14	15	17	18	19	20
3	Hypercube									
	de Bruijn									
	Cayley									
	DCC	512	1024	2048	4096	8192	16384	32768	65536	131072
	Two-diam	640	1280	2560	5120		20480	40960	73728	147456

Deg	Gra \ Diam	5	6	7	8	9	10	11	12	13
4	Hypercube									
	de Bruijn		120							
	Cayley			128	256	512	1024	2048	4096	8192
	DCC			512	1024	2048	8192	16384	32768	131072
	Two-diam		160	640	1296	2592	9216	20480	40960	147456
5	Hypercube	32								
	de Bruijn			720						
	Cayley									
	DCC	128	512	2048	8192	16384	65536	131072		
	Two-diam	160	640	2560	9216	18432	73728	147456		

Table 5.2: A comparison of graph size

5.3 Network Properties

Table 5.4 summaries these graphs by comparing their network properties.

Deg	Gra \ Diam	4	5	6	7	8	9	10
6	Hypercube			64				
	de Bruijn						5040	
	Cayley		243	729	2187	6561	19683	
	DCC		512	1024	8192	32768	131072	
	Two-diam		648	1792	9216	36864	147456	
7	Hypercube				128			
	de Bruijn							40320
	Cayley							
	DCC		1024	4096	16384	131072		
	Two-diam		1152	4608	18432			
8	Hypercube					256		
	de Bruijn							
	Cayley	256	1024	4096	16384	65536		
	DCC	512	2048	8192	65536	131072		
	Two-diam	576	2304	9216		147456		
9	Hypercube						512	
	de Bruijn							
	Cayley							
	DCC		2048	16384	131072			
	Two-diam		2304	18432				
10	Hypercube							1024
	de Bruijn							
	Cayley	625	3125	15625	78125			
	DCC	1024	4096	32768	131072			
	Two-diam	1152	4608	36864	147456			

Table 5.3: A comparison of graph size (continued)

Gra. / Pro	Hypercube	De Bruijn	Cayley	DCC Linear	2-dimensional
Regularity	Y	N	Y	Y	Y
Symmetry	Y	N	Y	N	N
Connectivity	Max	Max	Max	Max for even degree	Max for even degree
Extensibility	Y	Y	Y	Y	Y

Table 5.4: A comparison of topology

Chapter 6

Conclusion

6.1 Research Results

The design of networks is a very important topic. Networks are used in the Internet, distributed systems, and multicomputers to improve performance, resource utilization, and to provide users with a more convenient and reliable environment. The underlying network is one of the key factors in designing these systems. Choosing a proper topology is a crucial decision with respect to the design of a network.

The multidimensional linear congruential network model provides a fertile source to build a variety of topologies. The construction is started by choosing a set of generators and a graph size. The graph constructed varies with different generator sets and the properties of the graph are completely specified by its generator set. Properties of these graphs can be analyzed by algebraic tools, proved as a whole class, and inherited by new graphs. The idea of generalizing a linear congruential graph into a multidimensional linear congruential graph has been shown to be successful. DCC linear congruential graphs are much larger in size than de Bruijn graphs of the same degree and diameter. We have shown that with a proper choice of two-dimensional generators, they generate graphs that are even larger than DCC linear congruential graphs and therefore are larger than any other families of graphs.

In addition, we have proved that two-dimensional linear congruential graphs can retain those good network properties found in the DCC linear congruential graphs. We summarize these properties as below.

1. They are regular and are defined for both even and odd degree.
2. They are defined for many different graph sizes.
3. They allow increasing the number of vertices without increasing the degree.
4. A graph of higher degree contains a graph of lower degree as a proper subgraph.
5. A graph of degree 2k is 2k-connected; a graph of 2k+1 is at least 2k-connected.
6. Their structures are extensible.
7. They are constructed by a uniform method and most of the graphs of degree Δ and diameter D in the previous tables are larger in size than the largest known graphs of $(\Delta, D-1)$.
8. They can be decomposed into a number of edge-disjoint cycles.

Also, we have obtained some empirical results on the generators to generate 2^i cycles for $i \geq 0$, and sufficient conditions to generate edge-disjoint cycles in graphs of degree 4. In particular, the lemma to construct a Hamiltonian cycle in a two-dimensional graph generalizes the theorem of Knuth on the linear function having maximum cycle length [28]. Since linear functions having maximum cycle length are used in random number generation and random numbers have many important applications [28], these two-dimensional generators chosen by our lemma and having maximum cycle length could be used in the applications of random numbers.

6.2 Future Considerations

Three directions can be pointed out for future research work.

1. Looking for the complete understanding of the cycle structures of generators in the two-dimensional space.

2. Constructing a table of graphs in a d-dimensional space for $d \geq 3$ to further improve the diameters.

3. Looking for the complete understanding of the cycle structures of generators in the d-dimensional space for $d \geq 3$.

In the first direction, we have found a lemma to create a Hamiltonian cycle and a lemma to create edge-disjoint cycles in graphs of degree 4. Two topics for future research in the two-dimensional case are:

- Looking for a lemma to create 2^j cycles, $j \geq 1$ for generators of complex matrix form.

- Looking for a lemma to create edge-disjoint cycles of higher degrees.

If these lemmas are available in the future, it will be possible to create disjoint consecutive cycles in graphs of any degree in the two-dimensional space.

All graphs of our results use 2^i as the length of one dimension; we did not try other numbers as the base of the power function. Furthermore, given a DCC linear congruential graph of order p^i, where p is a number other than 2, the same Lemma 4.2.1 and Lemma 4.2.2 can tell us how to choose the constants of a generator to create disjoint consecutive cycles. We also do not have such a similar conclusion for those generators of complex matrix form.

In the second and third directions, it will be the same task as we did for the two-dimensional case. However, we believe that our methods of reasoning can be extended into the multidimensional case, so that some of the properties we discovered in the two-dimensional space can therefore be extended without much difficulty. Instead of using exhaustive search, it will be time-saving if the steps are made as below.

1. Establishing d-dimensional generators that create disjoint consecutive cycles by using simple matrix form.

2. Looking for lemmas that create disjoint consecutive cycles for generators of complex matrix form.

3. Searching for low-diameter graphs of given degree and size by constructing disjoint consecutive cycles.

Bibliography

[1] S.B. Akers, B. Krishnamurthy: *Group Graphs as Interconnection Networks*, the 14th International Conference on Fault Tolerant Computing, 1984, pp.422-427.

[2] S.B. Akers, B. Krishnamurthy: *On Group Graphs and Their Fault Tolerance*. IEEE Trans. on Computers, vol. c-36, July, 1987, pp.885-888.

[3] S.B. Akers, B. Krishnamurthy: *A Group-Theoretic Model for Symmetric Interconnection Networks*. IEEE Trans. on Computers, vol. c-38, April, 1989, pp.555-565.

[4] B. Arden, H. Lee: *Analysis of Chordal Ring Network*. IEEE Trans. on Computers, vol. c-30, April, 1981, pp.25-29.

[5] B. Arden, H. Lee: *A Regular Network for Multicomputer Systems*. IEEE Trans. on Computers. vol. c-31, Jan., 1982, pp.57-66.

[6] B.W. Arden, K.W. Tang: *Representations and Routing for Cayley Graphs*. IEEE Trans. on Communications, vol. 39, Nov., 1991, pp.1533-1537.

[7] W.C. Athas, C.L. Seitz: *Multicomputers: Message-Passing Concurrent Computers*. Computer, Aug., 1989, pp.9-24.

[8] J.C. Bermond, C. Delorme, J.J. Quisquater: *Strategies for Interconnection Network: some methods for graph theory*. Journal of Parallel and Distributed Computing, vol. 3, 1986, pp.443-449.

[9] J.C. Bermond, C. Peyrat: *de Bruijn and Kautz networks: a competitor for the hypercube.* Hypercube and Distributed Computers, 1989, North Holland, pp.279-294.

[10] B. Bollobas, W.F. de la Vega: *The Diameters of Random Graphs.* Combinatorica 2, 1982, pp.125-134.

[11] D.V. Chudnovsky, G.V. Chudnovsky, M.M. Denneau: *Regular Graphs with Small Diameter as Models for Interconnection Networks.* Proceedings of the Third International Conference on Supercomputing, 1988, pp.232-239.

[12] F.R.K. Chung: *Diameters of Graphs: Old Problems and New Results.* Congressus Numerantium 60, 1987, pp.295-317.

[13] D.E. Comer: *Internetworking With TCP/IP.* Prentice Hall, 1991.

[14] W.J. Dally: *A Fine-Grain, Message-Passing Processing Node.* Concurrent Computation, Plenum Press, 1988, pp.375-387.

[15] C. Delorme: *A Table of Large Graphs of Small Degrees and Diameters.* private communication, 1992.

[16] S. Dolinar, T.M. Ko, R. McEliece: *Some VLSI Decompositions of the de Bruijn Graph.* Discrete Mathematics, 106/107, 1992, North-Holland, pp.189-198.

[17] K. Doty: *New Design for Dense Processor Interconnection Networks.* IEEE Trans. on Computers, vol. c-33, May, 1984, pp.67-70.

[18] T.Y. Feng: *A Survey of Interconnection Networks.* Computer, Dec., 1981, pp.12-27.

[19] F. Harary: *Graph Theory.* Addison-Wesley, 1969.

[20] J.P. Hayes: *Architecture of a Hypercube Supercomputer.* Proceedings of 1986 International Conference on Parallel Processing, 1986, pp.653-660.

[21] L.S. Haynes, R.L. Lau, D.P. Siewiorek, D.W. Mizell: *A Survey of Highly Parallel Computing.* Computer, Jan., 1982, pp.9-24.

[22] W.D. Hillis: *The Connection Machine.* ACM distinguished dissertation, the MIT press, Cambridge, MA, 1989.

[23] R.W. Hockney. C.R. Jesshope: *Parallel Computer.* Adam Hilger, 1981.

[24] A. Hoffman, R. Singleton: *On Moore Graphs with diameters 2 and 3.* IBM Journal, Nov., 1960, pp.497-504.

[25] K. Hwang, F.A. Briggs: *Computer Architecture and Parallel Processing.* McGraw-Hill.

[26] M. Imase, M. Itoh: *Design to Minimize Diameter on building-Block Network.* IEEE Trans. on Computers, vol. c-30, June, 1981, pp.439-442.

[27] M.R. Jerrum, S. Skyum: *Families of Fixed Degree Graphs for Processor Interconnection.* IEEE Trans. on Computers, vol. c-33, Feb., 1984, pp.190-194.

[28] D.E. Knuth: *The Art of Computer Programming.* Seminumerical Algorithms, Addison-Wesley, vol. II, 1972.

[29] F.T. Leighton: *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.

[30] G. Memmi, Y. Raillard: *Some New Results About the (d,k) Graph Problem.* IEEE Trans. on Computers, vol. c-31, Aug., 1982, pp.71-78.

[31] J. Opatrny, D. Sotteau, N. Srinivasan, K. Thulasiraman: *DCC Linear Congruential Graphs: A New Class of Interconnection Network.* submitted to IEEE International Symposium on Circuits and Systems.

[32] J. Opatrny, C.C. Koung: *Two-dimensional Linear Congruential Graphs.* submitted to the 24th Southeastern International Conference on Combinatorics, Graph Theory, and Computing.

[33] J.C. Peterson et al.: *The Mark III Hypercube-Ensemble Concurrent Processor*, Prceedings of 1985 International Conference on Parallel Processing, pp.71-73, Aug. 1985.

[34] R.F. Rashid: *Threads of a New System.* Unix Review, Aug., 1986, pp.37-49.

[35] K.H. Rosen: *Discrete Mathematics and Its Applications.* The Random House, 1988.

[36] Y. Saad. M.H. Schultz: *Topological Properties of Hypercubes.* IEEE Trans. on Computers, vol. c-37, July, 1988, pp.867-872.

[37] M.R. Samantham, D.K. Pradhan: *The De Bruijn Multiprocess or Network: A Versatile Parallel Processing and Sorting Network for VLSI.* IEEE Trans. on Computers, vol. c-38, April, 1989, pp.567-581.

[38] R.R. Seban: *FTN Topology and Protocols.* Journal of Parallel and Distributed Computing, 1991, pp.51-62.

[39] C.L. Seitz: *The Cosmic Cube.* Communication of the ACM, vol. 28, Jan., 1985, pp.22-33.

[40] T. Soneoka, H. Nakada, M. Imase: *Design of a D-connected Digraph with a Minimum Number of Edges and Quasiminimal Diameter.* Networks, vol. 14, 1984, pp.63-74.

[41] R.M. Storwick: *Improved Construction Techniques for (d,k) Graphs.* IEEE Trans. on Computers, vol. c-19, Dec., 1970, pp.1214-1216.

[42] A.S. Tanenbaum, R.V. Renesse: *Distributed Operating System.* Computing Surveys, vol. 17, no.4, 1985, pp.419-467.