Routing and Broadcasting in Two-dimensional Linear Congruential Graphs of Degree Four

Kuo-Jui Raymond Lin

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 1994

ISBN 0-315-97642-X

Abstract

Routing and Broadcasting in Two-dimensional Linear Congruential Graphs of Degree Four

Kuo-Jui Raymond Lin

A two-dimensional linear congruential graph $G(\{f_1, f_2, \cdots, f_k\}, (s_1, s_2))$, or 2-D LC graph for short, of size $s_1 \times s_2$ is a graph in which the set of vertices is the set of pairs of integers $\{(x, y) \mid 0 \leq x < s_1, 0 \leq y < s_2\}$, and there is an edge from (x, y) to $f_i(x, y) \bmod (s_1, s_2)$ for any (x, y) and any function in the set $\{f_1, f_2, \cdots, f_k\}$ of two-dimensional linear functions. 2-D LC graphs were introduced in [1].

In this thesis, we consider the problems of routing and broadcasting in 2-D LC graphs of degree 4 in which f_1 generates a Hamiltonian cycle, and f_2 generates a few disjoint cycles. First, some symmetric properties of 2-D LC graphs are discussed. We then discuss a greedy global routing algorithm with a breadth first search scheme, which uses a large number of path tables, and provide a more efficient scheme having only one path table for all vertices of a graph. Two depth first distributed routing algorithms are proposed. In absence of faults, both algorithms route messages along the shortest path between a pair of vertices. They are evaluated in various cases of faults. We also discuss an algorithm for finding a set of disjoint paths between a pair of vertices. A broadcasting algorithm in LC graphs is proposed. We give functions for which the broadcasting can be done in time $O(\log_2 n)$, which is asymptotically optimal. We then investigate the broadcasting problem for 2-D LC graphs and propose strategies that improve the broadcast time.

Acknowledgement

First, I wish to express my sincere gratitude to my thesis supervisor, Dr. Jaroslav Opatrny, for his excellent guidance and kindly support during the course of this thesis. I am also grateful for his patience and encouragement.

I thank Mr. Ching-Chun Koung for his encouragement and friendship. His work on the properties of the network model has inspired ideas and his diligence has influenced me in more ways than he knows.

I appreciate my parents for their constant support, and finally I thank my wife for her consideration and support.

Dedicated to my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Problem Domain

Because of the advance in VLSI technology, the continuing dec.ine in the cost of microprocessors and the demand for reliable computing systems with high total computing power, many research efforts have been made to design massively parallel processors. One of these systems is the multicomputer system. A multicomputer system consists of a set of computers interconnected by a network in which a typical computer might consist of a main processor, a local memory and a communication processor to control the communications with other computers in the system. As the processing resources, control and data are distributed in these systems, they are more reliable than the systems with a single processor.

Many problems that cannot be solved efficiently by sequential computers can be divided into smaller tasks or subproblems that can be executed in parallel. Each of these tasks may be assigned to a processor. After a processor finished its task, the result may be sent to other processors. Communication among processors in such a network is achieved by a message passing protocol.

In the design of a massively parallel computers system, one of the important design decisions is the topology of the network interconnecting all computers in the system. Some basic considerations concerning the interconnection network are as follows. The number of processors must be large enough to meet the demand for high total computing power. The number of communication links at each computer must be small since the numbe^r^ of connection pins is limited. To maintain a low message latency, the distance (minimum number of links) between any two computers must be small.

In the design of computer networks, graphs are usually used to model the computer networks, in which the vertices correspond to the computers in the networks, and the edges correspond to the communication links. In graph theoretic terms, a graph corresponding to a network having the above desirable properties is a graph having large size (i.e., large number of vertices), small degree and small diameter. However, these three parameters are mutually related. For example, given a degree Δ and diameter D, the size S of a graph that can be constructed is bounded. It is so-called (Δ, D)-graph problem.

Moore provided an upper bound on the size of a graph with given degree and diameter as follows.

$$S(\Delta, D) = 1 + \Delta + \Delta(\Delta - 1) + \cdots + \Delta(\Delta - 1)^{D-1}$$

Many research efforts have been made to construct graphs of size near the upper bound. It has been proved that the complete graph ($D = 1$, any Δ), rings (any D, $\Delta = 2$), *Petersen graph* ($D = 2$, $\Delta = 3$) as well as *Hoffman and Singleton graph* ($D = 2$, $\Delta = 7$) attain the upper bound. However, for $D > 2$ and $\Delta > 2$, the Moore bound is unattainable [2, 3].

In addition to the above desirable properties of interconnection networks, some other issues on network performance are fault-tolerance, traffic congestion and ease of routing. A network with redundant paths between pairs of nodes can tolerate more faulty nodes or communication links. Therefore, the number of disjoint paths between any two vertices should be as large as possible. However, the maximum number of disjoint paths is bounded by the degree of a graph. A network that has its links uniformly distributed can be expected to minimize the congestion problem. The property of every vertex having the same number of incident edges (i.e., regular graph) is thus desirable. A symmetric topology of a network allows the use of identical processors at every node since every node plays an identical role in the network. The same routing algorithm can thus be applied to each node. It enables the design of low cost routing hardware.

Based on the above considerations, some graphs have been proposed as good candidates for interconnection networks su .h as hypercubes [4, 5, 6], de Bruijn graphs [7, 8, 9] and star graphs [10, 11, 12]. De Bruijn graphs and their variations are well known large graphs of a given degree and diameter. Recently, a new class of graphs, called (multidimensional) linear congruential graphs, has been proposed [13, 1]. A two-dimensional linear congruential graph $G(\{f_1, f_2, \cdots, f_k\}, (s_1, s_2))$ of size $s_1 \times s_2$ and degree $2k$ is a graph in which the set of vertices is the set of pairs of integers $\{(x, y) \mid 0 \leq x < s_1, 0 \leq y < s_2\}$ and there is an edge from (x, y) to $f_i(x, y) \bmod (s_1, s_2)$ for any (x, y) and any function in the set $\{f_1, f_2, \cdots, f_k\}$ of two-dimensional linear functions. It provides a uniform method to construct very large graphs for any fixed degree and size.

It has been shown that with a proper choice of functions, a two-dimensional linear congruential graph is larger than the de Bruijn graph of the same degree and diameter, and it is regular and maximally connected (i.e., the number of disjoint paths between any pair of vertices of a graph is equal to the degree of the graph). Moreover, we have proved that they have some symmetric properties. In other words, two-dimensional linear congruential graphs have the desirable properties of interconnection networks. Motivated by these properties, the aim of our research is to investigate the routing and broadcasting in two-dimensional linear congruential graphs, in particular, of degree four.

An efficient message routing scheme is one of the most important components of a multicomputer system. To decrease the message latency, sending messages along the shortest path between given source and destination nodes is expected. In general, there are two types of routing schemes, global routing and distributed routing, for sending a message from a source node to a destination node.

In a global routing scheme, the decision of a route for sending a message is made at the source node. There are two ways to decide a route. One way is to construct a table at each node. The table contains the shortest paths from the node to each other nodes. Whenever a node wants to send a message, the node simply checks the table to decide a route. This way is usually used when the calculation of a route is time-consuming. Another way is to re-calculate a route at a source node whenever a message will be sent by the source node.

In distributed routing scheme, a route for sending a message is not completely decided by the source node, but by each intermediate node traversed by the message.

Each intermediate node chooses a link that probably leads to a shortest path to the destination of the message to forward the message.

Since a multicomputer system may consist of a large number of nodes and communication links, it is possible that some nodes or links in the system fail. Therefore, a fault-tolerant routing scheme is required especially when a multicomputer system is used in a mission-critical application. It should be able to send messages as long as the source and destination nodes are connected. In global routing scheme, each node must have the knowledge of all faulty components (i.e., faulty nodes and links) since a route is completely decided at a source node. It is costly to maintain the knowledge. On the contrary, in distributed routing scheme, each node may only have to know the conditions of its incident links and adjacent nodes since the routing decisions are made at each intermediate node. However, due to the lack of global knowledge of faults, a path determined by this scheme may not be the shortest.

As we mentioned, the existence of multiple disjoint paths between pairs of nodes implies that a network can tolerate more faults. In addition, message congestion can be minimized since messages between the same pair of nodes can be sent along disjoint paths.

Broadcasting generalizes the routing process. It is a process of sending a message from a node to all other nodes in the network in which any node that already received the message can send a message to at most one of its neighbors in one time unit. Broadcasting is an important part of many parallel algorithms. Loading the same program code from a front end to all the processors in a multicomputer system is a typical example of broadcasting [14].

1.2 Thesis Outline

In the next chapter, two major categories of parallel processing: multicomputer and multiprocessor are introduced. Some basic notations of graph theory are reviewed. The performance and cost issues in the design of a network topology are discussed. Formal definitions of global routing, distributed routing, disjoint paths and broadcasting are given. We then review the routing and broadcasting algorithms for some well-known graphs: hypercubes, de Bruijn graphs and star graphs.

In Chapter 3, we review some definitions and properties of linear congruential graphs. Some symmetric properties of one- and two-dimensional linear congruential graphs of degree four are proved.

In Chapter 4, the problems of global routing, distributed routing, finding disjoint paths and broadcasting in two-dimensional linear congruential graphs of degree four are discussed.

A global routing algorithm using a breadth first search approach is proposed. The number of path tables used by this algorithm for a graph is proportional to the size of the graph. Therefore, a more efficient global routing algorithm is discussed. It only uses a unique path table and a very small size-independent mapping table for all vertices of a graph. Furthermore, the maximum length of any path determined by this algorithm is very close to the diameter of the graph.

Two distributed routing algorithms are investigated. They both use a depth first search approach and route messages along the shortest path in absence of faults in the network. Empirical results for these two algorithms applying in various faulty conditions are given.

An algorithm that can find the maximum number of disjoint paths for any pair of vertices is proposed. The lengths of the disjoint paths are bounded.

We propose a broadcasting algorithm that provides an upper bound $O(log_2 n)$ on broadcast time of certain one-dimensional linear congruential graphs, where n denotes the size of a graph. However, it does not generalize in a simple way to two-dimensional linear congruential graphs. The empirical broadcast times of two-dimensional linear congruential graphs are evaluated. The broadcasting in two-dimensional linear congruential graphs can be done in time similar to the time required for de Bruijn graphs of the same degree and comparable size. Three strategies to improve broadcast time are discussed.

Chapter 5 presents conclusions.

Chapter 2

Review of Current Network Models

2.1 Overview

The multiprocessor and multicomputer are two major categories of parallel processors. A *multiprocessor* system consists of a set of n identical processors and a global memory. The memory can be considered as if it were split into n "banks", and shared among the n processors. Communication among the processors is via shared memory. The primary advantage of multiprocessor systems is that the data access is transparent to the user, i.e., the user may consider the data as being kept in a large memory, which is accessible to any processor. The architecture makes the programming of the machine easy. However, it may lead to memory contention and thus degrade the performance of system.

A *multicomputer* system consists of a set of processors, each of which has its own local memory (hence each processor can be considered as a computer). There is no shared memory and no global synchronization. Communication among the processors is via message passing, and computation is data driven. The primary advantage of such architecture is the simplicity of its design. The nodes are identical or of a few different kinds. Thus the system can be constructed at relatively low cost.

Whether a parallel processors system is implemented by a shared memory multiprocessor or a message passing multicomputer, an efficient interconnection network to interconnect the communicating elements is important for the performance of the system.

Before we discuss the performance of interconnection networks according to the properties of their related graphs, some basic notations of graphs will be reviewed below [15, 16, 17].

2.1.1 Basic Notations of Graph Theory

A *graph* $G(V, E)$ consists of two sets: a finite set V of elements called *vertices* and a finite set E of elements called *edges*. Each edge is identified with a pair of vertices. If the edges of a graph G are identified with unordered pairs of vertices, then G is called an *undirected graph.*

The vertices u and v associated with an edge e are called the *end vertices* of e. The edge e is then denoted as $e = (u, v)$. All edges having the same pair of end vertices are called *parallel edges.* If $e = (v, v)$, then the edge e is called a *self-loop* at vertex v. A graph is called a *simple graph* if it has no parallel edges or self-loops. We will focus our attention on undirected simple graphs.

If there is an edge $e = (u, v)$, then we say that edge e is *incident* with vertex u (or v), and vertices u and v are *adjacent.* If two edges e_1 and e_2 have a common end vertex, edges e_1 and e_2 are said to be *adjacent.*

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

The *degree of a vertex* is the number of edges incident with the vertex. The *degree of a graph* G is the maximum degree among all vertices of G and is denoted by Δ. A Graph in which all vertices have the same degree is called *a regular graph.*

A *path* of *length* n from the vertex u to the vertex v is a sequence $e_1e_2\cdots e_n$ of edges together with a sequence $v_1v_2\cdots v_{n+1}$ of vertices where $e_i = (v_i, v_{i+1})$ for $1 \leq i \leq n$, and $v_1 = u$, $v_{n+1} = v$. We will usually denote the path as $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{n+1}$.

The *distance* between two vertices of a graph is the length of the shortest path between them. The *diameter* of a graph G is the maximum distance among all pairs of vertices of the graph and is denoted by $D(G)$.

A path with a sequence $v_1v_2\cdots v_{n+1}$ of vertices is *closed* if $v_1 = v_{n+1}$. If the vertices v_1, v_2, $\cdots$, v_n are all distinct, the closed path is called a *cycle.*

A path with a sequence $v_1v_2\cdots v_n$ of vertices is called a *Hamiltonian path* for a graph $G(V, E)$ if v_1, v_2, $\cdots$, v_n are all distinct and $\{v_1, v_2, \cdots, v_n\} = V$. A cycle $v_1 v_2 \cdots v_n v_1$ is called a *Hamiltonian cycle* if $v_1 v_2 \cdots v_n$ is a Hamiltonian path. A graph that has a Hamiltonian cycle is called a *Hamiltonian graph.*

A graph $G'(V', E')$ is a *subgraph* of a graph $G(V, E)$ if V' and E' are subsets of V and E, respectively, such that an edge (u, v) is in E' only if u and v are in V'.

A graph is called *connected* if every pair of distinct vertices is joined by a path. The removal of a vertex v from a graph G induces a subgraph of G that contains all vertices except v and the edges not incident with v. The *connectivity* of a graph is the minimum number of vertices that need to be removed to disconnect the graph.

A family of graphs $\{G_1, G_2, \cdots, G_i, \cdots\}$ is 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.

An *isomorphism* of a graph $G_1(V_1, E_1)$ onto a graph $G_2(V_2, E_2)$ is a one-to-one correspondence $\alpha : V_1 \rightarrow V_2$ such that (u, v) is in E_1 if and only if $(\alpha(u), \alpha(v))$ is in E_2. Two graphs G_1 and G_2 are *isomorphic*, written $G_1 \simeq G_2$, if there is an isomorphism α of one onto the other. An isomorphism of a graph onto itself is called an *automorphism* of the graph. A graph is *vertex-transitive* if and only if for every pair of vertices v_1 and v_2, there is an automorphism of the graph that maps v_1 into v_2. A graph is *edge-transitive* if and only if for every pair of edges e_1 and e_2, there is an automorphism of the graph that maps e_1 into e_2.

2.1.2 Network Topology, Routing and Broadcasting Algorithms

The interconnection topology and related routing algorithm are two major issues in the design of an interconnection network. In choosing a topology, two factors: the cost and performance should be carefully considered [18]. Since we use graphs to model interconnection networks, we will analyze the performance and cost issues in graph theoretic terms.

In order to achieve high performance within an interconnection network, the following properties of the corresponding graph are desirable.

- Small diameter: messages in a network may pass through a number of intermediate nodes before reaching their destinations. At each intermediate node, both the message processing overhead and queuing delay result in the communication latency experienced by a message. Small diameter of the graph means that few number of intermediate nodes are visited by every message. Thus the communication time is kept relatively short.

- Regular graph: If some of nodes in the network have less links than the others, it is likely for the network to have bottlenecks at these nodes. A regular graph has all of its edges uniformly distributed, therefore reducing the probability of the occurrence of bottlenecks.

- Large connectivity: If there are many possible paths between each pair of nodes in a network, in the presence of faults, the communication between non-faulty nodes may continue to work in a higher probability. High degree of connectivity prevents the non-faulty nodes from disconnected. (A graph with connectivity n can tolerate at least $n - 1$ faulty nodes.)

When considering the cost of a network, two important properties of the corresponding graph should be listed:

- Small degree: The number of connection pins is limited, and each physical connection costs money. For each vertex of a graph, if the number of edges incident with the vertex is small, i.e., the number of wires in the corresponding network is small, then the cost is likely to be low.

- Regular graph: If each node in a network has the same number of links, one router design can be used for all nodes.

Obviously, some of the above properties are conflicting. For example, a graph with small degree cannot have large connectivity. Thus any decision made in choosing a topology will experience a tradeoff between cost and performance.

In addition to choosing a topology for a network, we must also select an algorithm for transmitting messages through the network. This algorithm is called a routing algorithm. A *routing algorithm* is a set of rules that specifies, for each message, through which path to route the message and the data required to make such decision. We will focus our study on four algorithms: global routing, distributed routing, finding disjoint paths and broadcasting.

In *global routing*, a path for sending a message from a source node to a destination node in a network is decided at the source node. Each intermediate node traversed by the message just follows the path to forward the message. There may be more than one path between these two nodes. We usually choose one of the shortest paths to route messages. In the presence of faults, each node requires global knowledge of faults existing in the network to ensure that the chosen path is valid.

Obviously, if each node is equipped with the information on all faulty components (nodes or links), a source node can always determine a fault-free path for every message to its destination as long as these two nodes are connected. However, it is too costly in space and time for each node to maintain the information especially when the size of the network is large, or when this information keeps changing.

In *distributed routing*, each node is allowed to keep only the failure information essential for making correct routing decision. The information kept in each node may be only the conditions of its adjacent components, or some times, more information other than just the adjacent components. Here, we only consider the former case.

Under this scheme, when a node wants to send a message, it is responsible only to choose one of its non-faulty links to send the message according to some pre-defined rules. When an intermediate node receives the message, it chooses a link to forward the massage or returns the message to the sending node if necessary. Because each node does not keep the information on all faulty components in the network, the chosen paths may not be the shortest possible ones.

In a set of *disjoint paths* between two nodes u and v, any two paths do not traverse a common node except nodes u and v. The existence of disjoint paths between two nodes may speed up transferring large amount of data between these two nodes. It also provides a way to select alternative routes in case a given node in a path is faulty. The maximum number of disjoint paths between any two nodes depends on the connectivity of the graph used to model the network.

Broadcasting is an information dissemination process in which a single originator sends a message to all other nodes in a network as quickly as possible. There are several models of broadcasting. The most common are as follows.

1. If, for each node in a network, all its links share the same transmitter, then

each node can send a message through at most one of its links during each time unit.

2. If, for each node in a network, each link of the node has a dedicated transmitter, then each node can send messages through all its links simultaneously.

Here, we only discuss the former, i.e., a node that already received a message can only send the message to one of its adjacent nodes during each time unit. Many fast and efficient parallel algorithms require the use of broadcasting scheme as a basic step.

In the following sections, we will review three kinds of graphs: hypercubes, de Bruijn graphs, star graphs as well as the results of their related routing algorithms.

2.2 The Hypercube Graph

The hypercube is also known as cosmic cube, n-cube, binary n-cube and boolean n-cube. Due to its structural regularity for easy construction and high potential for the parallel execution of various algorithms [19], several multicomputer configurations have been designed based on its topology [18, 4, 20].

2.2.1 Definitions and Properties

DEFINITION 2.2.1 [19] r-Dimensional Hypercube

In an r-dimensional hypercube, *each vertex corresponds to an r-bit binary string, and two vertices are linked with an edge if and only if their binary strings differ in precisely one bit. Thus an r-dimensional hypercube has $N = 2^r$ vertices and $r2^{r-1}$ edges.*

The edges of a hypercube are partitioned according to the dimensions that they traverse. An edge is called a *dimension k edge* if it links two vertices that differ in the kth bit position.

Hypercubes are regular graphs since each vertex of an r-dimensional hypercube is adjacent to $r = \log_2 N$ other vertices, one for each bit position. Its degree is thus r, i.e., the degree increases logarithmically with its size.

It is proved in [19] that hypercubes have diameter $r = \log_2 N$ and are vertex- and edge-transitive.

The connectivity of an r-dimensional hypercube is r, since there are r disjoint paths between each two vertices [5]. We will discuss the scheme of finding disjoint paths later.

An r-dimensional hypercube can be constructed from two copies of the $(r-1)$-dimensional hypercube [19]. Thus hypercubes are recursive.

From the viewpoint of network performance, hypercubes have the following desirable properties: diameter $r = \log_2 N$, regular graph and large connectivity. However, when building a large hypercube, its large degree will increase the cost.

2.2.2 Routing and Broadcasting Algorithms

Global Routing

Given two vertices v_1 and v_2 of an r-dimensional hypercube (of size $N = 2^r$), one of the ways to find the shortest paths from v_1 to v_2 is to modify the bits of v_1 one at a time in order to transform the binary representation of v_1 into v_2. Each time one bit is changed, one edge will be crossed.

If the binary representations of v_1 and v_2 differ in k bits, $1 \leq k \leq r$, then the length of the shortest paths is k. Since the k different bits can be changed in any order, there are $k!$ shortest paths.

Distributed Routing

In a fault-free network modeled by a hypercube, each intermediate vertex can determine the next hop of a message by examining the destination of the message and choosing, from all its adjacent vertices, one that is closest to the destination. One possible method is to align the binary representation of the source vertex with that of the destination vertex from right to left bit-by-bit. However, this scheme becomes invalid in the presence of faulty components (faulty vertices or faulty edges) in the network. Thus a fault-tolerant distributed routing scheme is required.

In recent years, researchers have proposed several schemes on this subject. Some require each vertex to keep more informations on faults rather than just the conditions of its incident edges and adjacent vertices [21, 22]. We will here, however, narrow our discussion to the case in which each vertex keeps only the information on its incident edges and adjacent vertices.

A simple algorithm under this constraint was proposed in [21]. This algorithm can route messages between any pair of non-faulty vertices of an r-dimensional hypercube as long as the total number of faulty components is less than r.

The evaluation of the algorithm running on an r-dimensional hypercube is briefly summarized as follows.

Let f be the number of faulty edges, g be the number of faulty vertices, and $len(u,v)$ be the distance between two given vertices u and v.

- If $f+g < r$, and u,v are not faulty, then it requires at most $len(u,v)+2(f+g)$ hops to send a message from u to v.

- If there are $(r-1)$ faulty edges, i.e., $f = r-1$, then it will route a message from a vertex u to another vertex v via one of the shortest paths between u and v with probability greater than $(1-\frac{r_1(1-r_1^l)}{1-r_1})$ where $r_1 = \frac{r-1}{r2^{r-1}}$, and $l = len(u,v)$. For example, if there are 4 faulty edges in a 5-dimensional hypercube, and $len(u,v) = 5$, then the probability of routing via the shortest path is greater than 0.948.

- If there are $(r-1)$ faulty vertices, i.e., $g = r-1$, then it will route a message from a vertex u to another vertex v via one of the shortest paths between u and v with probability greater than $(1-\frac{(r-1)r_2(1-r_2^{l-1})}{(2^r-2)(1-r_2)})$ where $r_2 = \frac{r-2}{2^r-3}$, and $l = len(u,v)$. For example, if there are 4 faulty vertices in a 5-dimensional hypercube, and $len(u,v) = 5$, then the probability of routing via the shortest path is greater than 0.985.

If g or f is equal to $r-1$, the above two probabilities of routing via the shortest path increase with r. However, it does not guarantee that a message can be routed between any two non-faulty vertices if the number of faulty components $(f+g)$ is greater than or equal to r. Thus we now discuss another routing algorithm using a depth-first search approach [23].

Under the depth-first search routing scheme, each message contains a stack that keeps track of the vertices traversed by the message, and tries to avoid visiting a vertex more than once except when backtracking is forced (i.e., when there is no alternative path available). Thus the algorithm will always route a message to its destination successfully as long as the source and destination are connected.

Similarly to the previous routing scheme, we list some important results as follows.

- In the worst case, the algorithm uses $len(u,v) + 2(2^r - len(u,v) - 1)$ hops to send a message from a vertex u to a vertex v of an r-dimensional hypercube if u and v are connected.

- In an r-dimensional hypercube, it will route a message from a vertex u to another vertex v, $len(u,v) = r$, via one of the shortest paths between u and v with probability P greater than or equal to
  $(1 - \sum_{j=1}^{\min(r,f)} \frac{C_{f-j}^{r2^{r-1}-j}}{C_f^{r2^{r-1}}})$ if there are only f faulty edges, or
  $(1 - \sum_{j=2}^{\min(r,g)} \frac{C_{g-j}^{2^r-2-j}}{C_g^{2^r-2}})$ if there are only g faulty vertices.

  For example, if $r = 5$, and $f = 4$, then the probability of routing via the shortest path is $P \geq 0.948$.

  If $r = 5$, and $g = 4$, then the probability of routing via the shortest path is $P \geq 0.985$.

Actually, the probabilities of routing via the shortest path in the above two algorithms are very similar. However, the expected length $len(u,v) + 2(f+g)$ of a path obtained by the first algorithm is closer to the length of the shortest paths if $f + g < r$.

Finding Disjoint Paths

It has been proved in [5] that for each two vertices u and v of an r-dimensional hypercube, if the distance between u and v is l (i.e., the binary representations of u and v differ in l bits), then we can always find r disjoint paths between u and v, in which l paths are of length l, and $(r - l)$ paths are of length $(l + 2)$. For example, let $r = 5$, $u = 00000$, and $v = 11100$. Then the five disjoint paths are

00000 → 10000 → 11000 → 11100,

00000 → 01000 → 01100 → 11100,

$00000 \rightarrow 00100 \rightarrow 10100 \rightarrow 11100,$

$00000 \rightarrow 00010 \rightarrow 10010 \rightarrow 11010 \rightarrow 11110 \rightarrow 11100,$

$00000 \rightarrow 00001 \rightarrow 10001 \rightarrow 11001 \rightarrow 11101 \rightarrow 11100.$

Each path starts changing a bit in different position (i.e., crossing an edge in different dimension), and then changes the bits that are different from the corresponding bit of the destination one by one in the cyclically left (or right) order until the destination is reached.

Broadcasting

An r-dimensional hypercube can be constructed from two $(r-1)$-dimensional hypercubes by simply connecting the ith vertex of one $(r-1)$-dimensional hypercube to the ith vertex of another for $0 \leq i < \frac{N}{2}$ where $N = 2^r$. The edges that connect the two copies are dimension r edges. We can simply prove by induction that broadcasting from any vertex of an r-dimensional hypercube can be done in r steps.

We first consider a one-dimensional hypercube. Since there are only two vertices connected by a dimension one edge, one vertex can inform another in one step. Assume that broadcasting from any vertex of a $(k-1)$-dimensional hypercube can be done in $k-1$ steps. Thus in a k-dimensional hypercube, any vertex v can inform all the vertices that belong to the same copy of the $(k-1)$-dimensional hypercube containing v in $k-1$ steps. In step k, each vertex that has been informed can send the message along its dimension k edge, and therefore every vertex of the k-dimensional hypercube can be informed in k steps.

2.3 The de Bruijn Graph

2.3.1 Definitions and Properties

Similarly to a hypercube, the vertices of a de Bruijn graph are strings of characters, and the diameter of a de Bruijn graph grows logarithmically with its size. A de Bruijn digraph is defined as follows.

DEFINITION 2.3.1 [24] de Bruijn Digraph $B(d, D)$

Each vertex of a de Bruijn digraph $B(d, D)$ *corresponds to a string of length D on an alphabet A of size d where we usually denote $A = \{0, 1, 2, \cdots, d-1\}$. There is* *an arc (a directed edge) from each vertex $u_1u_2\cdots u_D$ to vertices $u_2u_3\cdots u_D\alpha$ where α is any letter of A. Therefore, $B(d,D)$ has d^D vertices and d^{D+1} directed edges.*

According to the above definition, in addition to having outdegree d, each vertex of the de Bruijn digraph $B(d,D)$ also has indegree d. Thus $B(d,D)$ is a regular digraph.

When $d = 2$, the labels of vertices of the de Bruijn digraph $B(d,D)$ are the same as those of the D-dimensional hypercube, and $B(2,D)$ is called a *D-dimensional binary de Bruijn graph.* Although the structure of $B(2,D)$ has little in common with the D-dimensional hypercube, many D-dimensional hypercube computations can be efficiently simulated on $B(2,D)$ [19].

Since we only consider undirected graphs, we present the definition of the undirected de Bruijn graph $UB(d,D)$ as follows.

DEFINITION 2.3.2 [7] Undirected de Bruijn Graph $UB(d,D)$

An undirected de Bruijn graph $UB(d,D)$ *is the undirected graph obtained from the de Bruijn digraph $B(d,D)$ by omitting the orientations of the edges, removing the self-loops, and replacing each double edge by a single edge. The edges of a vertex can be classified into two sets, left-shift edges and right-shift edges such that the vertex $u_1u_2\cdots u_D$ is connected to all vertices $u_2u_3\cdots u_D\alpha$ by its left-shift edges and $\alpha u_1u_2\cdots u_{D-1}$ by its right-shift edges where α is any letter of the alphabet A.*

Clearly, the diameter of the undirected de Bruijn graph $UB(d,D)$ is at most D since given a source vertex $u = u_1u_2\cdots u_D$ and a destination vertex $v = v_1v_2\cdots v_D$, we can change the label of the source vertex to be that of the destination vertex by continuously shifting the label of the source vertex either left or right, and we have to change at most D letters. Actually, the diameter of $UB(d,D)$ is exactly D since it requires D shifts to change the label $00\cdots 0$ to the label $11\cdots 1$ A shifting operation corresponds to an edge traversed. Thus a path from u to v can be given by either $v = u_1u_2\cdots u_D \rightarrow u_2u_3\cdots u_Dv_1 \rightarrow u_3\cdots u_Dv_1v_2 \rightarrow \cdots \rightarrow v_1v_2\cdots v_D = v$ or $u = u_1u_2\cdots u_D \rightarrow v_Du_1u_2\cdots u_{D-1} \rightarrow v_{D-1}v_Du_1\cdots u_{D-2} \rightarrow \cdots \rightarrow v_1v_2\cdots v_D = v$. The above routing scheme is very easy, but the resulting paths may not be the shortest. We will discuss an algorithm that determines the shortest paths for any two vertices of $UB(d,D)$ in the later subsection.

An undirected de Bruijn graph $UB(d, D)$ contains $d^D - d^2$ vertices of degree $2d$, d vertices of degree $2d - 2$ and $d^2 - d$ vertices of degree $2d - 1$ [25]. Thus $UB(d, D)$ is of degree $2d$, but it is not a regular graph.

The connectivity of $UB(d, D)$ is $2d - 2$ [26]. $UB(d, D)$ is maximally connected since the minimum degree of any vertex of $UB(d, D)$ is also $2d - 2$.

There are two methods to construct a $B(d, D)$ from a smaller de Bruijn graph. Both are more complex than that of hypercubes. The first method is to maintain the degree of the original graph constant and to increase its diameter by 1, i.e., $B(d, D)$ can be constructed from $B(d, D-1)$ by line graph operation [7]. The second method is to maintain the diameter of the original graph constant and to increase its degree by 2, i.e., $B(d, D)$ can be constructed from d copies of $B(d-1, D)$ [8]. The advantages of the second method are that $B(d - 1, D)$ is a subgraph of $B(d, D)$, and that the method can also be used for undirected de Bruijn graphs.

Besides the low diameter $\log_d N$ (where $N = d^D$ is the size of $UB(d, D)$) and maximum connectivity, the undirected de Bruijn graphs have the following advantages over the hypercubes.

- The degree and diameter of the undirected de Bruijn graphs are not related. For a given size d^D of graph, we can choose either a small degree or a small diameter depending on the requirement of the design. However, a hypercube always has its degree equal to its diameter.

- For a given degree $\Delta = 2d$ and a given diameter D, the size of $UB(d, D)$ is d^D. Comparing with the D-dimensional hypercube (of size 2^D, degree and diameter D), if we let $\Delta = D$, then the size of $UB(\frac{\Delta}{2}, D)$ is $(\frac{\Delta}{2})^D$, which is much greater than 2^D when $\Delta > 4$.

- We can construct a de Bruijn graph $UB(d, D)$ of any size d^D. However, the size of a hypercube must be a power of 2.

On the contrary, undirected de Bruijn graphs have the following disadvantages.

- Undirected de Bruijn graphs are defined only for even degree. However, a similar graph has been defined for any odd degree [7].

- Undirected de Bruijn graphs are not regular.

2.3.2 Routing and Broadcasting Algorithms

Global Routing

The algorithm described in the previous subsection generates two paths of length D between any two vertices u and v of $UB(d,D)$. These two paths may not be the shortest. For example, let $u = u_1u_2\cdots u_D$, $v = v_1v_2\cdots v_D$, and $u_i\cdots u_D = v_1\cdots v_{D-i+1}$ where $1 < i \leq D$. Then the path $u = u_1u_2\cdots u_i\cdots u_D \rightarrow u_2u_3\cdots u_i \cdots u_Dv_{D-i+2} \rightarrow u_3\cdots u_i\cdots u_Dv_{D-i+2}v_{D-i+3} \rightarrow \cdots \rightarrow u_iu_{i+1}\cdots u_Dv_{D-i+2}\cdots v_D \rightarrow v_1v_2\cdots v_D = v$ is of length $i-1 < D$.

However, this scheme is still not suitable for some pairs of vertices in finding the shortest paths since the edges traversed by the paths are all left-shift edges (or all right-shift edges). For example, let $D = 10$, $u = u_1u_2\cdots u_{10}$, $v = v_1v_2\cdots v_{10}$. If $u_2\cdots u_6 = v_5\cdots v_9$ are the only consecutively identical strings in u and v, and $u_1 \neq v_{10}$, $u_{10} \neq v_1$ then the paths found by the above scheme are of length $D = 10$.

An algorithm proposed in [27] is always able to find the shortest paths between any two vertices u and v of $UB(d,D)$, in which a path may contain both left- and right-shift edges. According to the algorithm, the shortest paths in the above case are given as follows. $u = u_1u_2\cdots u_{10} \rightarrow u_2u_3\cdots u_{10}w_1 \rightarrow v_4u_2u_3\cdots u_{10} \rightarrow v_3v_4u_2\cdots u_9 \rightarrow v_2v_3v_4u_2\cdots u_8 \rightarrow v_1\cdots v_4u_2\cdots u_7 \rightarrow w_2v_1\cdots v_4u_2\cdots u_6 \rightarrow v_1\cdots v_4u_2 \cdots u_6v_{10} = v$ where w_1 and w_2 are any letters in A.

We can observe that it is able to find more than one shortest path of length $7 < D$ between u and v, since w_1 and w_2 are any letters in A. Thus the communications could be more or less balanced.

Finding Disjoint Paths

The connectivity of $UB(d,D)$ is $2d-2$, consequently, there are at least $2d-2$ disjoint paths between any two vertices of $UB(d,D)$.

For any two vertices $u = u_1u_2\cdots u_D$ and $v = v_1v_2\cdots v_D$ of $UB(d,D)$, we can easily find d disjoint paths between them of length at most $2D$ as follows [26]. $P = \{p_i \mid u = u_1u_2\cdots u_D \rightarrow u_2u_3\cdots u_Di \rightarrow u_3u_4\cdots u_Dii \rightarrow \cdots \rightarrow ii\cdots i \rightarrow v_Dii\cdots i \rightarrow v_{D-1}v_Dii\cdots i \rightarrow \cdots \rightarrow v_2v_3\cdots v_Di \rightarrow v_1v_2\cdots v_D = v, \text{for } 0 \leq i \leq d-1\}$. If any two vertices in a path p_i are identical, i.e., there is a cycle in the path, then we can remove the cycle from the path to get a shorter path.

One of the disadvantages of the scheme is that, for each pair of vertices, the set of disjoint paths between them always traverses the vertices $00\cdots 0$, $11\cdots 1$, $\cdots$, $(d-1)(d-1)\cdots(d-1)$. It may lead to traffic congestion at these vertices.

Another algorithm can find $d-1$ disjoint paths of length at most $D+1$ between any two vertices u and v [7]. The set of paths is $P = \{p_i \mid u = u_1u_2\cdots u_D \rightarrow u_2u_3\cdots u_Di \rightarrow u_3u_4\cdots u_Div_1 \rightarrow u_4u_5\cdots u_Div_1v_2 \rightarrow \cdots \rightarrow iv_1\cdots v_{D-2}v_{D-1} \rightarrow v_1\cdots v_{D-1}v_D = v$, for $0 \leq i \leq d-1\}$. There are d paths in P, but two of them may conflict. For example, let $u = 0000$ and $v = 1111$ be two vertices of $UB(d,4)$. Then both p_0 and p_1 contain vertex 0001. Similarly, any path in P that contains cycles can become a shorter path. Therefore, there are $d-1$ disjoint paths of length at most $D+1$.

Broadcasting

Broadcasting algorithms for de Bruijn graphs have been proposed in [28]. They provide upper bounds on the broadcast time of both directed and undirected de Bruijn graphs as follows. For de Bruijn digraphs, the broadcast time $b(B(d,D)) \leq \frac{d+1}{2}D + \frac{d}{2}$, and for undirected de Bruijn graph, the broadcast time $b(UB(d,D)) \leq \frac{d+1}{2}D + \frac{d+1}{2}$. These two upper bounds are very similar.

Two important properties have been proved in [29] as follows.

- For any $p \leq d$, there is a spanning directed p-ary tree of depth at most $D\lceil\log_p d\rceil$ in a de Bruijn digraph $B(d,D)$. Thus $b(B(d,D)) \leq pD\lceil\log_p d\rceil$.

- The broadcast time $b(B(pq,D))$ can be established in terms of $b(B(p,D))$ and $b(B(q,D))$.

According to the above properties, the upper bounds on broadcast time of de Bruijn digraphs are improved for $d \geq 15$. Since the undirected de Bruijn graph can be obtained from the de Bruijn digraph by removing the restriction of the orientations of the edges, an upper bound on $b(B(d,D))$ can also be considered as the upper bound on $b(UB(d,D))$. Since the upper bounds on $b(B(d,D))$ and $b(UB(d,D))$ proposed in [28] are very similar, the results in [29] also improve the upper bound on $b(UB(d,D))$ for some d.

2.4 The Star Graph

Like the hypercube, the star graph possesses rich structure, symmetric properties, and fault-tolerant capabilities. However, it has a smaller diameter and degree than the hypercube of similar size.

2.4.1 Definitions and Properties

The star graph is a member of a class of graphs called Cayley graphs. This class of graphs uses a group-theoretic approach as a basis for defining graphs. Actually, some properties of the star graph are generic properties of the Cayley graph (e.g., every Cayley graph is vertex-transitive [10]). Thus, before reviewing the definition of the star graph, we first review the definition of the Cayley graph below.

DEFINITION 2.4.1 [10] Cayley Graph

Let G be a finite group consisting of the set of finite permutations generated by a set of generators $g = \{g_1, g_2, \cdots, g_n\}$. A Cayley graph *is a graph in which the vertices correspond to the elements of the group G, and the edges correspond to the action of the generators on vertices. That is, there is an edge from a vertex u to a vertex v if and only if there is a generator $g_i \in g$ such that $ug_i = v$ in the group G.*

Here, the set g of generators must be closed under inverse so that the resulting graph can be viewed as being undirected.

The star graph and pancake graph are well known examples of Cayley graphs. In addition, hypercubes and cube-connected cycles can be defined as Cayley graphs [10]. In this thesis, we focus our attention only on the star graph. The definition of the star graph can be given as follows.

DEFINITION 2.4.2 [30] n-Star Graph S_n

*An n-*star graph *denoted by S_n is the Cayley graph specified by a set g of generators on a group G of all permutations on n symbols, and g is defined as follows. The set g consists of $n-1$ transpositions $g_2, g_3, \cdots, g_n$ where $g_i \in g$ is the transposition that interchanges the symbol in the ith position of a vertex with the symbol in its first position, and leaves the remaining symbols in the same position. That* *is, g consists of the following generators:* $g_2 = (2134 \cdots n)$, $g_3 = (3214 \cdots n)$, $\cdots$, $g_n = (n234 \cdots (n-1)1)$.

Since there are $n!$ permutations on n symbols, the n-star graph S_n consists of $n!$ vertices. There are $n-1$ generators applied on every vertex, and there is no self-loop at any vertex. Therefore, S_n is of degree $n-1$, and it is a regular graph.

The connectivity of S_n is $n-1$ [31]. Since its degree is also $n-1$, the star graph is maximally connected. The diameter $D(S_n)$ is $\lfloor \frac{3}{2}(n-1) \rfloor$ [10]. Further discussion about the connectivity and diameter will be given in the following subsection.

It is proved in [10], due to the inherent properties of Cayley graphs, that the star graph is vertex- and edge-transitive as well as hierarchical. The hierarchical property is especially essential to the algorithm for finding disjoint paths and broadcasting, which will be introduced later.

From here on, we will use $\{A, B, C, \cdots, Z\}$ to denote the n symbols, and $ABC \cdots Z$ to denote the identity permutation I on n symbols. Z is not necessary the 26th symbol, but rather the nth symbol.

An n-star graph S_n on a group G can be decomposed in two ways [11]. An example is given in Figure 2.1.

The vertices of S_n can be decomposed into n subgroups of G based on the symbol in the last position. Each of the n subgroups (subgraphs) contains $(n-1)!$ permutations (vertices) and is isomorphic to S_{n-1}. These subgraphs are interconnected by edges corresponding to interchanging the symbol in the first position with the symbol in the last position. That is, any two subgraphs are directly connected by $\frac{(n-1)!}{n-1} = (n-2)!$ edges.

We will use X_i to denote the subgraph consisting of the vertices that contain the symbol X in the ith position. S_n can thus be decomposed into n subgraphs $A_n, B_n, \cdots, Z_n$. In addition, S_n can also be decomposed into n subgraphs, each isomorphic to S_{n-1}, along any one of its dimensions. For example, S_4 can be partitioned into four interconnected copies of S_3, denoted by A_4, B_4, C_4, D_4, or A_3, B_3, C_3, D_3, or A_2, B_2, C_2, D_2.

In addition to the above decomposition strategy, an n-star graph S_n can also be orthogonally decomposed into $A_1, A_2, \cdots, A_n$. Similarly, the subgraphs $A_2, A_3, \cdots, A_n$

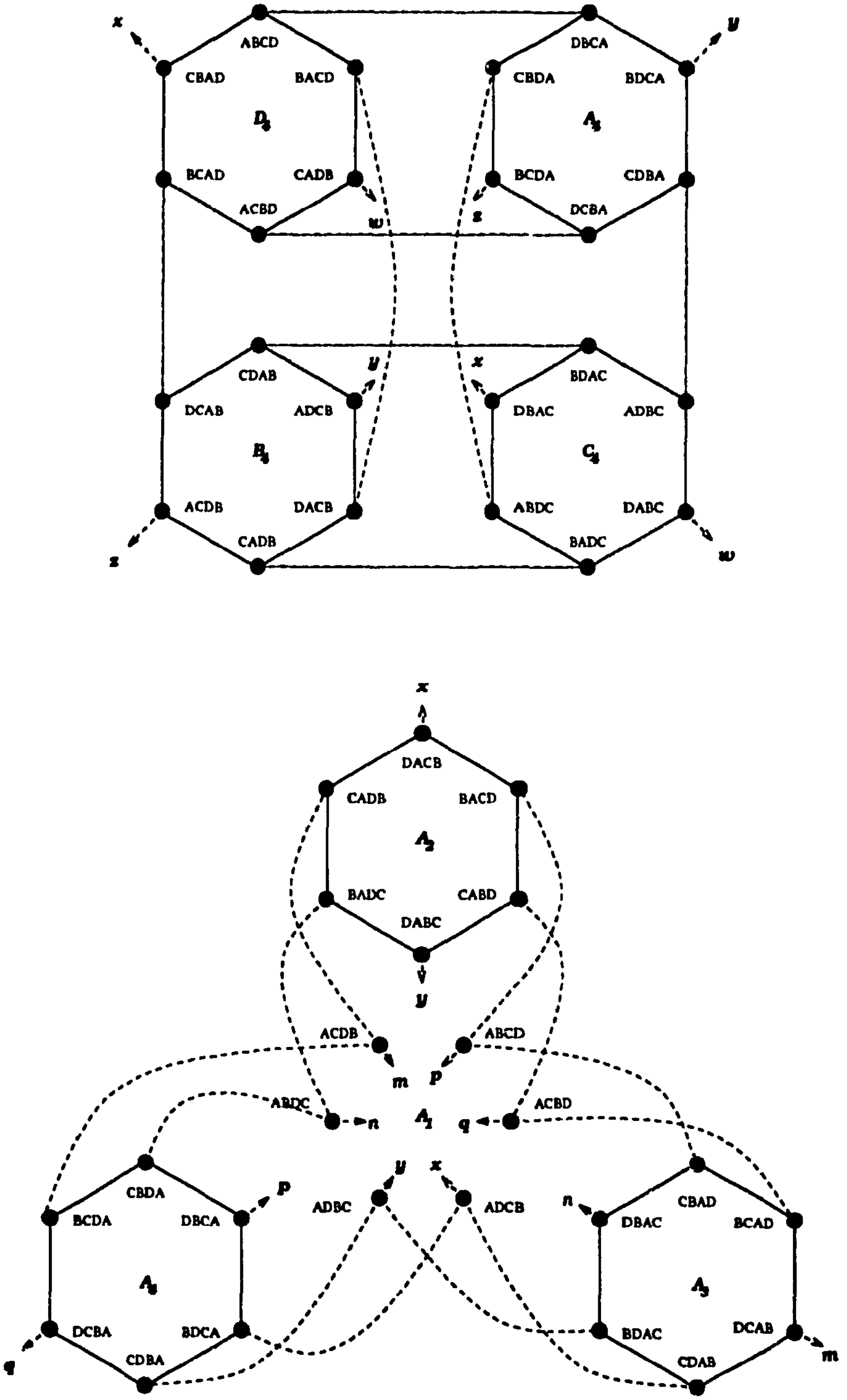


Figure 2.1: Two ways of decomposing a 4-star graph

are each isomorphic to S_{n-1}. However, A_1 is a collection of $(n-1)!$ isolated vertices. Every vertex in A_1 is connected to a vertex in each of the subgraphs $A_2, A_3, \cdots, A_n$. There is no edge between any two subgraphs A_i and A_j where $i \neq j$, and $2 \leq i, j \leq n$. Similarly, this orthogonal decomposition can be carried out using any symbol. For example, S_4 can be partitioned into A_1, A_2, A_3, A_4, or B_1, B_2, B_3, B_4, and so on.

Both the star graph and the hypercube are vertex- and edge-transitive, hierarchical, and maximally fault-tolerant. However, a star graph has smaller degree and diameter than the hypercube of comparable size. A table of comparison between their degrees and diameters can be found in [11].

We now discuss the routing and broadcasting algorithms on the star graph based on its symmetric and hierarchical properties.

2.4.2 Routing and Broadcasting Algorithms

Since the star graph is vertex-transitive, finding a path between two vertices is equivalent to finding a path between a specific vertex and the identity vertex.

Let $g_1, g_2, \cdots, g_p$ be a path from a vertex u to a vertex v of an n-star graph where $g_i \in g$. Since $ug_1, g_2, \cdots, g_p = v$, $v^{-1}ug_1, g_2, \cdots, g_p = v^{-1}v = I$. Therefore, $g_1, g_2, \cdots, g_p$ is also a path from the vertex $v^{-1}u$ to the identity vertex I. Consequently, the problem of routing becomes that of sorting. For example, if $u = BCAD$, and $v = CADB$, then the sequence $g_4g_3g_2$ is a path from u to v, and it is also a path from $v^{-1}u = (BDAC)(BCAD) = DABC$ to $I = ABCD$.

Global Routing

A simple rule for finding the shortest path from a vertex $v \neq I$ to the identity vertex $I = ABC \cdots Z$ of S_n can be introduced as follows [11].

Starting from v, repeat the following steps until I is reached.

1. If A is in the first position, move it to any position not occupied by the correct symbol.

2. Otherwise, (i.e., X, any symbol other than A, is in the first position) move X to the its correct position.

Actually, any vertex (permutation) v can be viewed as a set of cycles, i.e., cyclically order sets of symbols with the property that each symbol's desired position is that occupied by the next symbol in the same cycle. For example, if $v = BDIAGFEHC$, the set of cycles is then $(BDA)(IC)(GE)(F)(H)$. The cycles of length at least two are called *p-cycles* (e.g., (BDA), (IC), (GE)). The cycles of length one are called *i-cycles* (e.g., (F), (H)).

For each p-cycle, we must move the first symbol of the cycle to the first position of the vertex notation (i.e., interchange with A), and thus take one step.

We then consider the first p-cycle. If A is not in the first position of v, the above step is saved, and A must be the last symbol of the first p-cycle. Let the length of the first p-cycle be l. After $l - 1$ transpositions, A is in the first position. Thus another step is saved.

Consequently, the minimum distance $d(v)$ between v and I will be as follows.

$$d(v) = c + m - \begin{cases} 0 & \text{if } A \text{ is the first symbol of } v \\ 2 & \text{otherwise} \end{cases}$$

where c is the number of p-cycles, and m is the total number of symbols in p-cycles.

Therefore, in an n-star graph S_n, the maximum distance between any vertex and I is $\lfloor \frac{3}{2}(n-1) \rfloor$, the diameter is then also $\lfloor \frac{3}{2}(n-1) \rfloor$. The proof for the diameter of S_n can be found in [10]. For example,

1. when n is odd, if $v = A\underbrace{CB}\underbrace{ED}\cdots\underbrace{ZY}$, $d(v) = c + m = \frac{n-1}{2} + (n-1) = \lfloor \frac{3}{2}(n-1) \rfloor$.

2. when n is even, if $v = \underbrace{BA}\underbrace{DC}\cdots\underbrace{ZY}$, $d(v) = c+m-2 = \frac{n}{2}+n-2 = \lfloor \frac{3}{2}(n-1) \rfloor$. If $v = A\underbrace{DBC}\underbrace{FE}\underbrace{HG}\cdots\underbrace{ZY}$, $d(v) = c + m = \frac{n-2}{2} + (n-1) = \lfloor \frac{3}{2}(n-1) \rfloor$.

Distributed Routing

We now introduce a distributed routing algorithm presented in [32]. This routing algorithm is based on the greedy routing algorithm and uses the depth first search approach combined with a backtracking technique. The routing decision for a message is made at each intermediate vertex v according to the following information:

- the p-cycles and i-cycles of v,

- the state of its incident edges,
- the vertices that have been visited by the message and the path followed, which are carried by the message.

The routing algorithm running on an n-star graph S_n will route messages along one of the shortest paths from the source vertex u to the destination vertex v if no faults are encountered. In the presence of faults, it will always find a path from u to v with a bounded number $2(n!-1)-l$ of message hops if u and v are connected where l is the distance between u and v. Otherwise it will return the message to u after at most $2(n!-2)$ message hops.

At an intermediate vertex w, if all its edges that may lead to the shortest path from w to the destination vertex are invalid, and if vertex w can still forward the message along a valid edge, the number of extra added message hops due to this detour does not exceed four. A *valid edge* is defined as a non faulty edge that sends a message to a non visited vertex.

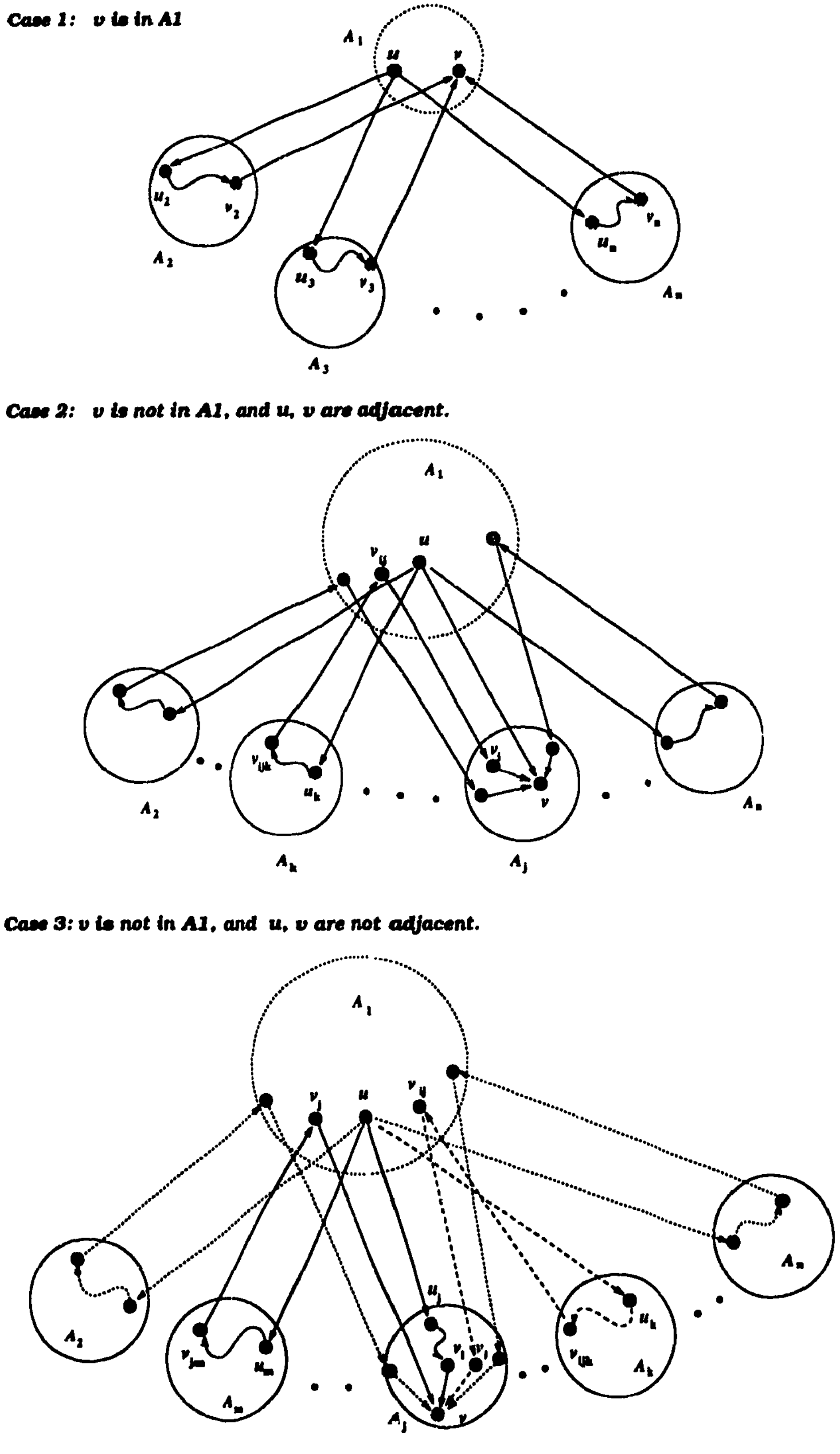
Finding Disjoint Paths

Since the connectivity of an n-star graph S_n is $n-1$, there are $n-1$ disjoint paths between any pair of vertices of S_n. We now discuss an algorithm for finding a set of $n-1$ disjoint paths between any two vertices of S_n using the orthogonal decomposition property of the star graph (i.e., S_n is decomposed into $A_1, A_2, \cdots, A_n$). Each of the disjoint paths is of length at most $D(S_n)+2$ if n is odd, or $D(S_n)+3$ if n is even ($D(S_n)$ is the diameter of S_n).

Since S_n is vertex-transitive, finding a set of disjoint paths between any two vertices of S_n is equivalent to finding a set of disjoint paths between $I \in A_1$ and a specific vertex v. We then can partition this problem into two cases as follows based on the subgraph where v is. (refer to Figure 2.2)

Let $u = I$, $g = \{g_2, g_3, \cdots, g_n\}$ be the set of generators, u_i be the vertex adjacent to u via the edge g_i (i.e., $ug_i = u_i$), and v_i be the vertex adjacent to v via the edge g_i (i.e., $vg_i = v_i$), $2 \leq i \leq n$.

Recall that every vertex in A_1 is connected to a vertex in each of the $n-1$ substars $A_2, A_3, \cdots, A_n$. If v is in A_1 (i.e., both u and v are in A_1), both u_i and v_i



Case 1: v is in A1

Case 2: v is not in A1, and u, v are adjacent.

Case 3: v is not in A1, and u, v are not adjacent.

Figure 2.2: $n - 1$ disjoint paths between two vertices of S_n

are in A_i. Obviously, there are $n-1$ disjoint paths $p_2, p_2, \cdots, p_n$ between u and v where p_i is the path (u, u_i, p'_i, v_i, v), and p'_i is the shortest path from u_i to v_i through A_i.

Since A_i is isomorphic to S_{n-1}, the distance between u_i and v_i in A_i is at most $D(S_{n-1})$. Thus, the length of any path p_i in the set of disjoint paths is at most $1 + D(S_{n-1}) + 1 = D(S_{n-1}) + 2$.

We now consider the cases that v is in A_j, $j \neq 1$. That is, the vertex v_i is in A_j if $i \neq j$, and v_j is in A_1.

If $v_j = u$, u and v are adjacent via the edge g_j, There is a path of length 1. Recall that any two vertices in the same substar A_j, $j \neq 1$, are connected to different vertices in A_1. Let $v_{ij} = v_i g_j$. Thus, v_{ij} is in A_1 if $i \neq j$. Since each vertex u_k is in different substar A_k $(2 \leq k \leq n)$, we can choose a one-to-one mapping between the set of v_{ij}'s and the set of A_k's where $i \neq j$, $k \neq j$, $2 \leq i, k \leq n$. Thus, the remaining $n-2$ disjoint paths are of the form

$$(u, u_k \in A_k, \cdots, v_{ijk} \in A_k, v_{ij} \in A_1, v_i \in A_j, v).$$

Each path is of length at most $D(S_{n-1}) + 4$.

If $v_j \neq u$, v and u are not adjacent. Since u_j is in A_j, we can choose a vertex v_l from v_i's such that one of the shortest paths between v and u_j traverses through v_l. Thus, there is a path of the form

$$(u, u_j \in A_j, \cdots, v_l \in A_j, v).$$

Its length is at most $D(S_{n-1}) + 1$. Another one of the disjoint paths can be of the form

$$(u, u_m \in A_m, \cdots, v_{jm} \in A_m, v_j \in A_1, v),$$

where $m \neq j$. Its length is at most $D(S_{n-1}) + 3$. We then choose a one-to-one mapping between the set of v_{ij}'s and the set of A_k's where $i \neq j$, $i \neq l$, $k \neq j$, $k \neq m$, $2 \leq i, k \leq n$. Similarly, the remaining $n-3$ disjoint paths are of the form

$$(u, u_k \in A_k, \cdots, v_{ijk} \in A_k, v_{ij} \in A_1, v_i \in A_j, v)$$

Each path is of length at most $D(S_{n-1}) + 4$.

If n is even, $D(S_n) = D(S_{n-1}) + 1$, else $D(S_n) = D(S_{n-1}) + 2$. Therefore, for any two vertices of S_n, there are $n-1$ disjoint paths, each of which is of length at most $D(S_{n-1}) + 4 = D(S_n) + 3$ if n is even, or $D(S_{n-1}) + 4 = D(S_n) + 2$ if n is odd.

Broadcasting

We now introduce a broadcasting scheme on star graphs, which was presented in [30]. Like the broadcasting scheme on hypercubes, this broadcasting scheme on star graphs employs the hierarchical structure of the star graphs. However, they start broadcasting a message in different order.

The broadcasting algorithm on an r-dimensional hypercube first broadcasts the message to vertices of the one-dimensional hypercube (subgraph) to which the originating vertex belongs, and then to all vertices of its two-dimensional hypercube (subgraph), and so on.

On the contrary, the broadcasting algorithm on an n-star S_n first sends the message to at least one vertex of each subgraph S_{n-1}, and then each subgraph S_{n-1} sends the message to at least one vertex of each its subgraph S_{n-2}, and so on. Consequently, the broadcasting problem on an n-star graph is reduced to n parallel broadcasting problem on $(n-1)$-star graphs.

Since an n-star graph S_n is vertex-transitive, we can assume that the originating vertex is the identity vertex $ABC \cdots Z$, which is in the substar Z_n isomorphic to S_{n-1}.

The algorithm of broadcasting a message to the other $n-1$ substars $A_n, B_n, \cdots, Y_n$ (each isomorphic to S_{n-1}) consists of two phases. It first routes the message to $n-1$ vertices of Z_n in $\lceil \log_2 n \rceil$ steps such that each of the $(n-1)$-stars $A_n, B_n, \cdots, Y_n$ is connected to one of the vertices. In the second phase, by applying g_n simultaneously, each of the $n-1$ vertices of Z_n that have received the message will route the message to a unique substar in one step.

For example, when the identity vertex $ABCDEFGH$ of S_8 broadcasts a message, the message is routed as follows.

- **In phase 1,**

  at step 1, $ABCDEFGH \xrightarrow{g_2} BACDEFGH.$

at step 2, $ABCDEFGH \xrightarrow{g_3} CBADEFGH,$

$BACDEFGH \xrightarrow{g_4} DACBEFGH.$

at step 3, $ABCDEFGH \xrightarrow{g_5} EBCDAFGH,$

$BACDEFGH \xrightarrow{g_6} FACDEBGH,$

$CBADEFGH \xrightarrow{g_7} GBADEFCH.$

- **In phase 2,** these seven vertices of the substar H_8 route the message simultaneously to the other seven substars $A_8, B_8, \cdots, G_8$ along its g_8 edge.

Then each of the n substars $A_n, B_n, \cdots, Z_n$ broadcasts the message in a similar way until 1-stars (consisting of only one vertex) are reached. Therefore, the broadcast time $b(S_n)$ is $\sum_{i=2}^{n}(\lceil \log_2 n \rceil + 1)$. Since the size of S_n is $N = n! \leq n^n$, and $b(S_n) = \sum_{i=2}^{n}(\lceil \log_2 n \rceil + 1) = O(n \log_2 n) = O(\log_2 N)$, this algorithm is a near optimal broadcasting algorithm.

Chapter 3

Symmetries in Two-Dimensional Linear Congruential Graphs

In this chapter, we first review the definitions and some properties of DCC linear congruential graphs and two-dimensional linear congruential graphs. We then prove the symmetric properties of two-dimensional linear congruential graphs $G(F, (2^i, s_2))$.

3.1 Definitions

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

We will first introduce the definition of linear function of dimension d.

DEFINITION 3.1.1 Linear Function of Dimension d

Let $d \in N$. We say 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.*

For example, $f(\vec{x}) = \vec{x}A + \vec{b}$ is a two-dimensional linear function where $A = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix}$, $\vec{b} = (3, 1)$, and $\vec{x} = (x, y)$ is a variable vector.

DEFINITION 3.1.2 d-Dimensional Linear Function of Simple Form and Function of Complex Form

Let $f(\vec{x}) = \vec{x}A + \vec{b}$ *be a d-dimensional linear function where*

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{pmatrix}.$$

We say the function f is of simple form *if* $a_{ij} = 0$ *for* $i \neq j, 1 \leq i,j \leq d$. *The* function f is of complex form *if* $a_{ij} \neq 0$, $i \neq j$ *for some* i,j, $1 \leq i,j \leq d$.

DEFINITION 3.1.3 Multidimensional Linear Congruential Graph

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

We use $G(F, \vec{s})$ to generate graphs of even degree, and $G(F, \vec{s}, g, V_1)$ to generate graphs of odd degree. The size of the graph is given by $\vec{s}$ and is equal to $s_1 \times s_2 \times \cdots \times s_d$. The linear functions in F and $F \cup \{g\}$ are called the *generators* of $G(F, \vec{s})$ and $G(F, \vec{s}, g, V_1)$, respectively.

The edges could be considered to be directed from a vertex $\vec{x} \in V$ to the vertex $f_i(\vec{x}) \bmod \vec{s}$, $1 \leq i \leq k$, but in this thesis, we will only consider the undirected case.

For example, $G(F, \vec{s})$ is a two-dimensional linear congruential graph on the vertex set $V = Z_{s_1} \times Z_{s_2}$ where $F = \{f_1, f_2\}$, $f_1(\vec{x}) = \vec{x}\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$ and $f_2(\vec{x}) = \vec{x}\begin{pmatrix} 9 & 0 \\ 1 & 1 \end{pmatrix} + (2,4)$ are two-dimensional linear functions, and $\vec{s} = (32, 9)$.

If $d = 1$, then the above definition corresponds to the linear congruential graph from [13]. In this case, all generators are ordinary linear functions.

DEFINITION 3.1.4 *d*-Dimensional Linear Congruential Graph of Simple Form and Graph of Complex Form

Let $G(F, \vec{s})$ be a d-dimensional linear congruential graph where $F = \{f_i(\vec{x}) \mid f_i(\vec{x}) = \vec{x}A_i + \vec{b_i},\ 1 \leq i \leq k$ for some $k\}$. We say the graph $G(F, \vec{s})$ is of simple form *if every d-dimensional linear functions f_i in F is of simple form. The* graph $G(F, \vec{s})$ is of complex form *if at least one of the d-dimensional linear functions f_i in F is of complex form.*

3.2 Review of Properties of Linear Congruential Graphs

In order to p₁ ove some symmetric properties of two-dimensional linear congruential graphs $G(\{f_1, f_\cdot\}, (2^i, s_2))$ of degree 4, we will describe some important properties of linear congruenιial graphs of dimension one and two, which are taken from the papers [13] and [1]. These papers contain detailed proofs of these Lemmas.

LEMMA 3.2.1 [13] 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 3.2.2 [13] Disjoint Cycles of Equal Lengths

Let n be a positive integer such that $n = k^i m$ for some integer $k > 1$, $i \geq 2$ 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 $\frac{n}{k^j}$ on the set $\{0, 1, \ldots, n-1\}$. The vertex sets of these cycles are $A_{1,j} = \{0, k^j, \ldots, n - k^j\}$, $A_{2,j} = \{1, k^j + 1, \ldots, n - k^j + 1\}$, $\cdots$, $A_{k^j,j} = \{k^j - 1, 2k^j - 1, \ldots, n - 1\}$. Furthermore, there is an edge between x_1 and x_2 in the graph generated by f_j only if $\mid x_2 - x_1 \mid$ is divisible by k^j but not by k^{j+1}.

LEMMA 3.2.3 **[1] Hamiltonian Cycle in a Two-dimensional Graph** $G(\{f\},(2^i,4k+1))$

Let $f(\vec{x}) = \vec{x}A + \vec{b}$ be a generator of a two-dimensional linear congruential graph $G(F,(2^i,4k+1))$ where k is an integer, $A = \begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix}$, and $\vec{b} = (b_1,b_2)$. If $a_{11}x + b_1$ generates a Hamiltonian cycle in $\{0,1,\ldots,2^i-1\}$, i.e., a_{11} and b_1 satisfy Lemma 3.2.1, and $\gcd(b_2,4k+1) = 1$, *then $f(\vec{x})$ also generates a Hamiltonian cycle in G.*

DEFINITION 3.2.1 **[13, 1] Extension of a Graph**

The extension *of a linear congruential graph $G(F,2^i)$ is defined to be the graph $G(F,2^{i+1})$. The* extension *of a two-dimensional linear congruential graph $G(F,(2^i,s_2))$ is defined to be the graph $G(F,(2^{i+1},s_2))$.*

LEMMA 3.2.4 **[1] Regular Cycle Structure in** $G(\{f\},(2^i,4k+1))$

Let $G_1 = G(\{f\},(2^i,4k+1))$ be a two-dimensional linear congruential graph where k is an integer, and $f(\vec{x}) = \vec{x}A + \vec{b}$, and let G_2 be the extension of G_1. Let $A = \begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix}$, $\vec{b} = (b_1,b_2)$. If $a_{11}x + b_1$ generates a Hamiltonian cycle in $\{0,1,\ldots,2^i-1\}$, i.e., a_{11} and b_1 satisfy Lemma 3.2.1, and $\gcd(b_2,4k+1) = 1$, *then $f(\vec{x})$ will generate the same cycle structure in both G_2 and G_1.*

The function f described as above is a regular generator and generates a Hamiltonian cycle in a two-dimensional graph $G(\{f\},(2^i,4k+1))$.

DEFINITION 3.2.2 **[13, 1] Edge-Change of a Graph**

Let $G(\{f\},2^i)$ be a linear congruential graph. We say that a vertex x of G has an edge-change when G is extended *if* $f(x) \bmod 2^i \neq f(x) \bmod 2^{i+1}$.

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

LEMMA 3.2.5 **[13] Symmetry of Extension in** $G(\{f\},2^i)$

Let $G_1 = G(\{f\},2^i)$ be a linear congruential graph where $f(x) = ax + b$, and let G_2 be the extension of G_1. If a is odd, then for every edge $(x_1,x_2) \in E(G_1)$,

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

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

LEMMA 3.2.6 [1] Symmetry of Extension in $G(\{f\},(2^i,s_2))$

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

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 $E(G_2)$.*

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 $E(G_2)$.*

3.3 Symmetric Properties of Two-Dimensional Linear Congruential Graphs $G(\{f_1,f_2\}, (2^i,4k+1))$

In this section, we will discuss the symmetric properties of two-dimensional graphs $G(\{f_1, f_2\}, (2^i,s_2))$ of degree 4 where $f_1(\vec{x}) = \vec{x} \begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1,b_2)$, $f_2(\vec{x}) = \vec{x} \begin{pmatrix} c_{11} & 0 \\ c_{21} & 1 \end{pmatrix} + (d_1,d_2)$, a_{11} and b_1 satisfy lemma 3.2.1, c_{11} and d_1 satisfy lemma 3.2.2, $s_2 = 4k+1$ for some integer k, $\gcd(b_2,s_2) = 1$, $\gcd(d_2,s_2) = 1$, and we will focus on $a_{11} = 5$, and $c_{11} = 9$.

Since $G(\{f_1,f_2\},(2^i,s_2))$ is a two-dimensional graph, we can express the variable vector $\vec{x}$ as (x,y) where x and y are variables, and $0 \le x < 2^i, 0 \le y < s_2$.

When $a_{21} = 0$ and $c_{21} = 0$, both of the generators are of simple form. Therefore, the graph is of simple form. When $a_{21} \neq 0$ or $c_{21} \neq 0$, at least one of the generators is of complex form, and the graph is of complex form. We will discuss these two cases, separately, in the following sections.

3.3.1 Graphs of Simple Form

In a graph $G(\{f_1, f_2\}, (2^i, s_2))$ of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} a_{11} & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, and $f_2(x,y) = (x,y)\begin{pmatrix} c_{11} & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, each of the generators f_1 and f_2 can be rewritten as two linear functions in which each one contains exactly one variable. For example, we can define $f_{1_x}(x) = a_{11}x + b_1$, $f_{1_y}(y) = y + b_2$ and $f_1(x,y) = (f_{1_x}(x), f_{1_y}(y))$.

Thus the graph $G(\{f_1, f_2\}, (2^i, s_2))$ can be obtained by the product of graphs $G_x = G(\{f_{1_x}, f_{2_x}\}, 2^i)$ and $G_y = G(\{f_{1_y}, f_{2_y}\}, s_2)$. The product of $G_x \circ G_y$ is defined as a graph with vertex set $V = V(G_x) \times V(G_y)$ and edge set $E = \{((u_x, u_y), (v_x, v_y)) \mid (u_x, v_x) \in E(G_x)$, $(u_y, v_y) \in E(G_y)$, and $v_x = f_{j_x}(u_x) \bmod 2^i$, $v_y = f_{k_y}(u_y) \bmod s_2$, for $j = k, 1 \leq j, k \leq 2$ $\}$.

In order to describe the symmetric properties of graphs, we will first review the definition of vertex transitivity.

An automorphism $\alpha : V(G) \rightarrow V(G)$ of a graph G is a one-to-one mapping such that $(u,v) \in E(G)$ if and only if $(\alpha(u), \alpha(v)) \in E(G)$.

DEFINITION 3.3.1 Vertex Transitivity

A graph G is vertex-transitive, *if for any two vertices $u, v \in V(G)$, there is an automorphism α of G such that $\alpha(u) = v$.*

The linear congruential graphs are not vertex-transitive in general, However, we will show that they have a restrictive type of symmetry with respect to a partition. Now, we will review the relationship among equivalence relations, equivalence classes and partitions. The proofs can be found in [15].

A *relation R* on a set S is called an *equivalence relation* if it is reflexive, symmetric and transitive. For example, the relation *congruence modulo m* $= \{(a,b) \mid a \bmod m = b \bmod m,\ a, b \in N\}$, $m \in N$, is an equivalence relation on N.

The set of all elements that are related to an element a by an equivalence relation R is called the *equivalence class* of a with respect to R and is denoted by $[a]_R$. The equivalence classes of the relation congruence modulo m are called the *congruence classes modulo m*. The congruence class of an integer a modulo m is denoted by $[a]_m$ and is equal to $\{b \mid (b - a) \bmod m = 0\}$.

A *partition* of S is a collection of disjoint nonempty subsets of S that have S as their union. The equivalence classes of an equivalence relation R are either equal or disjoint, and the union of these classes is all of S. Thus the equivalence classes form a partition of S. For example, the congruence classes modulo m on N : $[0]_m = \{0, m, 2m, \cdots\}$, $[1]_m = \{1, m+1, 2m+1, \cdots\}$, $\cdots$, $[m-1]_m = \{m-1, 2m-1, 3m-1, \cdots\}$ form a partition of N.

With the above-mentioned introduction on vertex transitivity and partition, we will now define a restrictive type of vertex transitivity, and we call it ***vertex transitivity with respect to a partition.***

DEFINITION 3.3.2 Vertex Transitivity with respect to a Partition P

A graph G is vertex-transitive with respect to a partition *P, if for any two vertices u, v in the same set in partition P of $V(G)$, there is an automorphism α of G such that $\alpha(u) = v$.*

Let $G(\{f_1, f_2\}, 2^i)$ be a one-dimensional linear congruential graph where $f_1(x) = 5x + b_1$, $f_2(x) = 9x + d_1$, b_1 satisfies Lemma 3.2.1, i.e., $\gcd(b_1, 2^i) = 1$, d_1 satisfies Lemma 3.2.2, i.e., $d_1 = 2\beta$, and $\gcd(\beta, 2^i) = 1$. We will prove that $G(\{f_1, f_2\}, 2^i)$ is vertex-transitive with respect to a partition P of $V(G)$ where P consists of the congruence classes modulo 2^{i-4} on $V(G)$: $[0]_{2^{i-4}}, [1]_{2^{i-4}}, \cdots, [2^{i-4}-1]_{2^{i-4}}$, i.e., there are 2^{i-4} congruence classes, and each of them consists of exactly 16 vertices.

THEOREM 3.3.1 Vertex Transitivity of a One-dimensional Graph G

Let $G(\{f_1, f_2\}, s)$ be a one-dimensional linear congruential graph where $f_1(y) = y + b_2$, $f_2(y) = y + d_2$, $\gcd(b_2, s) = 1$, and $\gcd(d_2, s) = 1$. Then G is vertex-transitive.

Proof:

Since $\gcd(b_2, s) = 1$, f_1 generates a Hamiltonian cycle. Let y_0 be a vertex in $V(G)$, and let n be an integer such that $f_2(y_0) \bmod s = f_1^n(y_0) \bmod s$. Thus

$$\begin{aligned} f_2(y_0) \bmod s &= (y_0 + d_2) \bmod s \\ f_1^n(y_0) \bmod s &= (y_0 + n \times b_2) \bmod s \end{aligned}$$

$$\Longrightarrow (n \times b_2) \bmod s = d_2 \bmod s$$

For every vertex $y \in V(G)$, $0 \leq y \leq s-1$, $y \neq y_0$,

$$\begin{aligned} f_1^n(y) \bmod s &= (y + n \times b_2) \bmod s \\ &= (y + d_2) \bmod s \\ &= f_2(y) \bmod s \end{aligned}$$

Thus there exists an integer n, $1 \leq n \leq s-1$, such that for every vertex $y \in V(G)$, $f_2(y) \bmod s = f_1^n(y) \bmod s$.

Let u, v be any two vertices in $V(G)$. Since $\gcd(b_2, s) = 1$, f_1 generates a Hamiltonian cycle. There is an integer m, $1 \leq m < s$, such that $f_1^m(u) \bmod s = v$.

Define $\alpha(y) = f_1^m(y) \bmod s$. Clearly, $\alpha(y)$ is a one-to-one mapping of $V(G)$ to $V(G)$, and $\alpha(u) = f_1^m(u) \bmod s = v$.

Assume (y_0, y_1) is an edge in $E(G)$, then

Case 1: $y_1 = f_1(y_0) \bmod s$

$$\begin{aligned} \alpha(y_0) &= f_1^m(y_0) \bmod s \\ \alpha(y_1) &= \alpha(f_1(y_0) \bmod s) = f_1^m(f_1(y_0) \bmod s) \bmod s \\ &= f_1(f_1^m(y_0) \bmod s) \bmod s = f_1(\alpha(y_0)) \bmod s \end{aligned}$$

So $(\alpha(y_0), \alpha(y_1))$ is also an edge in $E(G)$.

Case 2: $y_1 = f_2(y_0) \bmod s$

$$\begin{aligned} \alpha(y_0) &= f_1^m(y_0) \bmod s \\ \alpha(y_1) &= \alpha(f_2(y_0) \bmod s) = f_1^m(f_2(y_0) \bmod s) \bmod s \\ &= f_1^m(f_1^n(y_0) \bmod s) \bmod s = f_1^n(f_1^m(y_0) \bmod s) \bmod s \\ &= f_1^n(\alpha(y_0)) \bmod s = f_2(\alpha(y_0)) \bmod s \end{aligned}$$

So $(\alpha(y_0), \alpha(y_1))$ is also an edge in $E(G)$.

Thus α is an automorphism of G, i.e., for any two vertices u, v in $V(G)$, there is an automorphism such that $\alpha(u) = v$. G is vertex-transitive.

□

THEOREM 3.3.2 Vertex Transitivity of a One-dimensional Graph $G(\{f_1, f_2\}, 16)$

Let $G(\{f_1, f_2\}, 16)$ be a one-dimensional linear congruential graph where

(1) ... $f_1(x) = 5x + b_1$,

(2) ... $f_2(x) = 9x + d_1$,

(3) ... b_1 satisfies Lemma 3.2.1, i.e., $\gcd(b_1, 16) = 1$,

(4) ... d_1 satisfies Lemma 3.2.2, i.e., $d_1 = 2\beta$, and $\gcd(\beta, 16) = 1$.

There exists an integer n such that for every vertex $x \in V(G) = \{0, 1, \ldots, 15\}$, $f_2(x) \bmod 16 = f_1^n(x) \bmod 16$. *Furthermore, G is vertex-transitive.*

Proof:

Assume $x_0, x_1, \ldots, x_n, x_{n+1}, \ldots$ is a linear congruential sequence created by f_1. If $f_2(x) \bmod 16 = f_1^n(x) \bmod 16$, then

$$\begin{aligned} x_{n+1} &= f_2(x_1) \bmod 16 = f_2(f_1(x_0)) \bmod 16 \\ &= f_2(5x_0 + b_1) \bmod 16 = (9 * (5x_0 + b_1) + d_1) \bmod 16 \\ &= (45x_0 + 9b_1 + d_1) \bmod 16 \end{aligned}$$

$$\begin{aligned} x_{n+1} &= f_1(x_n) \bmod 16 = f_1(f_2(x_0)) \bmod 16 \\ &= f_1((9x_0 + d_1)) \bmod 16 = (5 * (9x_0 + d_1) + b_1) \bmod 16 \\ &= (45x_0 + b_1 + 5d_1) \bmod 16 \end{aligned}$$

According to the above two equations.

$$(9b_1 + d_1) \bmod 16 = (b_1 + 5d_1) \bmod 16$$

$$(8b_1) \bmod 16 = (4d_1) \bmod 16$$

$$8 * (b_1 \bmod 2) = 4 * (d_1 \bmod 4)$$

$$2 * (b_1 \bmod 2) = d_1 \bmod 4$$

Because $\gcd(b_1, 16) = 1$, $b_1 = 2\alpha + 1$ where $\alpha \in N$. Thus

$$2 * (b_1 \bmod 2) = 2 * ((2\alpha + 1) \bmod 2) = 2 * 1 = 2.$$

Because $d_1 = 2\beta$, and $\gcd(\beta, 16) = 1$, $d_1 = 2(2m+1)$ where $m \in N$. Thus

$$d_1 \bmod 4 = 2(2m+1) \bmod 4 = (4m+2) \bmod 4 = 2.$$

Therefore, for all b_1 and d_1, $2 * (b_1 \bmod 2) = d_1 \bmod 4$. That is, for every vertex $x \in V(G)$, there exists an integer n such that $f_2(x) \bmod 16 = f_1^n(x) \bmod 16$.

Since f_1 generates a Hamiltonian cycle, we can use a similar way to prove that G is vertex-transitive.

□

THEOREM 3.3.3 *Let $G(\{f_1, f_2\}, 16)$ be a one-dimensional linear congruential graph where f_1 and f_2 satisfy the conditions (1), (2), (3) and (4) in Theorem 3.3.2. There exists an integer $n = 2 + 4k$, $k \in N$, such that for every vertex $x \in V(G)$,* $f_2(x) \bmod 2^i = f_1^n(x) \bmod 2^i$.

Proof:

Let n be the integer from Theorem 3.3.2. For every $x \in V(G)$,

$$\begin{aligned} f_1(x) &= (5x + b_1) \bmod 16 \\ f_1^2(x) &= (5(5x + b_1) + b_1) \bmod 16 = (5^2 x + (5+1)b_1) \bmod 16 \\ &\vdots \\ f_1^n(x) &= (5^n x + (5^{n-1} + 5^{n-2} + \cdots + 5 + 1)) \bmod 16 \\ f_2(x) &= (9x + d_1) \bmod 16 \end{aligned}$$

For every $x \in V(G)$, $f_1^n(x) \bmod 16 = f_2(x) \bmod 16$, so $(5^n \bmod 16 = 9)$ must hold.

By observing the sequence of consecutive powers of 5 modulo 16:

$$(5^0, 5^1, 5^2, 5^3, 5^4, 5^5, 5^6, 5^7, 5^8, \ldots) \bmod 16 = (1, 5, 9, 13, 1, 5, 9, 13, 1, \ldots),$$

we see that we obtain value 9 only when $n = 2, 6, \ldots, 2 + 4k$. Thus

$$5^n \bmod 16 = 5^{2+4k} \bmod 16 = ((5^4)^k * 5^2) \bmod 16 = 5^2 = 9.$$

□

THEOREM 3.3.4 *Let $G(\{f_1, f_2\}, 2^i)$ be a one-dimensional linear congruential graph where f_1 and f_2 satisfy the conditions (1), (2), (3) and (4) in Theorem 3.3.2, and $i \geq 4$. For each $x \in V(G)$, there exists an integer $n_x = 2 + 4k_1 + 16k_2$ where $0 \leq k_1 \leq 3$, $k_2 \in N$, such that $f_2(x) \bmod 2^i = f_1^{n_x}(x) \bmod 2^i$.*

Proof:

To simplify the notation, we use n to denote n_x.

According to Theorem 3.3.2 and 3.3.3, $G(\{f_1, f_2\}, 16)$ is vertex-transitive, and for every vertex x of the graph, there exists an integer $n = 2 + 4k_1, 0 \leq k_1 \leq 3$, such that $f_2(x) \bmod 16 = f_1^n(x) \bmod 16$.

Let $G(\{f_1, f_2\}, 32)$ be the extension of $G(\{f_1, f_2\}, 16)$. According to Lemma 3.2.5, if (u, v) is an edge of $G(\{f_1, f_2\}, 16)$, then either (u, v) or $(u, v + 16)$ is an edge of $G(\{f_1, f_2\}, 32)$.

Let (u, v) be an edge constructed by f_2 in $G(\{f_1, f_2\}, 16)$. If

1. (u, v) is an edge constructed by f_2 in $G(\{f_1, f_2\}, 32)$, i.e., there is no edge-change, and
   (a) if the number of edge-changes formed by f_1 between u and v is even, then $v = f_2(u) \bmod 32 = f_1^n(u) \bmod 32$
   (b) if the number of edge-changes formed by f_1 between u and v is odd, then $v = f_2(u) \bmod 32 = f_1^{n+16}(u) \bmod 32$
2. $(u, v + 16)$ is an edge constructed by f_2 in $G(\{f_1, f_2\}, 32)$, i.e., there is an edge-change, and
   (a) if the number of edge-changes formed by f_1 between u and v is even, then $v + 16 = f_2(u) \bmod 32 = f_1^{n+16}(u) \bmod 32$
   (b) if the number of edge-changes formed by f_1 between u and v is odd, then $v + 16 = f_2(u) \bmod 32 = f_1^n(u) \bmod 32$

Similarly, let $G(\{f_1, f_2\}, 64)$ be the extension of $G(\{f_1, f_2\}, 32)$, we can derive that $f_2(u)$ must be equal to one of following:

$$f_1^n(u), f_1^{n+16}(u), f_1^{n+32}(u), f_1^{n+48}(u).$$

In a similar way, we can prove that for each vertex x of $G(\{f_1, f_2\}, 2^i)$, there exists an integer $n_2 = n + 16k_2 = 2 + 4k_1 + 16k_2$ where $0 \leq k_1 \leq 3$, and $k_2 \in N$, such that $f_2(x) \bmod 2^i = f_1^{n_2}(x) \bmod 2^i$.

□

THEOREM 3.3.5 *Let $G(\{f_1, f_2\}, 2^i)$ be a one-dimensional linear congruential graph where f_1 and f_2 satisfy the conditions (1), (2), (3) and (4) in Theorem 3.3.2, and $i \geq 4$. Let x be a vertex in $V(G)$, and let $[c]_{2^{i-4}}$ be the congruence class modulo 2^{i-4} such that x is in $[c]_{2^{i-4}}$, i.e., $x = m2^{i-4} + c$, $0 \leq m < 16$, and $0 \leq c < 2^{i-4}$. Then $x' = f_1^{k2^{i-4}}(x) \bmod 2^i$ is also in the class $[c]_{2^{i-4}}$ where $0 < k < 16$, i.e., $x' = m'2^{i-4} + c$, $0 \leq m' < 16$.*

Proof:

Assume $x_0, x_1, \ldots, x_{2^{i-4}}, \ldots$ is a linear congruential sequence created by f_1 in G, and let $x_0 = m2^{i-4} + c$. Thus

$$\begin{aligned}
x_1 &= f_1(x_0) \bmod 2^i = (5(m2^{i-4} + c) + b_1) \bmod 2^i \\
&= (5m2^{i-4} + 5c + b_1) \bmod 2^i \\
x_2 &= f_1(x_1) \bmod 2^i = (5(5m2^{i-4} + 5c + b_1) + b_1) \bmod 2^i \\
&= (5^2 m2^{i-4} + 5^2 c + (5+1)b_1) \bmod 2^i \\
&\vdots \\
x_{2^{i-4}} &= f_1(x_{2^{i-4}-1}) \bmod 2^i \\
&= (5^{2^{i-4}} m2^{i-4} + 5^{2^{i-4}} c + (5^{2^{i-4}-1} + 5^{2^{i-4}-2} + \cdots + 5 + 1)b_1) \bmod 2^i
\end{aligned}$$

To simplify the equation, we let $n = 2^{i-4}$. Thus

$$\begin{aligned}
x_n &= (5^n mn + 5^n c + (5^{n-1} + 5^{n-2} + \cdots + 5 + 1)b_1) \bmod 2^i \\
x_n \bmod 2^{i-4} &= (5^n mn + 5^n c + (5^{n-1} + 5^{n-2} + \cdots + 5 + 1)b_1) \bmod 2^{i-4} \\
&= (5^n mn + 5^n c + (5^{n-1} + 5^{n-2} + \cdots + 5 + 1)b_1) \bmod n \\
&= (5^n c + (5^{n-1} + 5^{n-2} + \cdots + 5 + 1)b_1) \bmod n
\end{aligned}$$

Now, we will complete the proof by induction.

Basis:

Let $i = 4$. Then $n = 2^{i-4} = 2^0 = 1$.

$$f_1^1(x_0) = x_1 \bmod 1 = x_0 \bmod 1 = 0 = c$$

Let $i = 5$. Then $n = 2^{i-4} = 2^1 = 2$.

$$f_1^2(x_0) = x_2 \bmod 2 = (5^2c + (5+1)b_1) \bmod 2 = (25c + 6b_1) \bmod 2 = c$$

Induction Step:

Assume when $i = i_1$ (i.e., $n = 2^{i_1-4}$), $x_n \bmod n = c$. Thus

$$(5^n c + (5^{n-1} + 5^{n-2} + \cdots + 5 + 1)b_1) \bmod n = c.$$

$$5^n \bmod n = 1, \text{ and } (5^{n-1} + 5^{n-2} + \cdots + 5 + 1) \bmod n = 0.$$

$$5^n = \alpha n + 1, \text{ and } (5^{n-1} + 5^{n-2} + \cdots + 5 + 1) = \beta n.$$

When $i = i_1 + 1$, then $n' = 2^{i-4} = 2^{i_1+1-4} = 2 * 2^{i_1-4} = 2n$.

$(5^{n'}c + (5^{n'-1} + 5^{n'-2} + \cdots + 5 + 1)b_1) \bmod n'$

$= (5^{2n}c + (5^{2n-1} + 5^{2n-2} + \cdots + 5 + 1)b_1) \bmod 2n$

$= (5^{2n}c + (5^{2n-1} + 5^{2n-2} + \cdots + 5^n + 5^{n-1} + \cdots + 5 + 1)b_1) \bmod 2n$

$= ((\alpha n + 1)^2 c + (5^n(5^{n-1} + \cdots + 5 + 1) + (5^{n-1} + \cdots + 5 + 1))b_1) \bmod 2n$

$= ((\alpha^2 n^2 + 2\alpha n + 1)c + (5^n + 1)(5^{n-1} + \cdots + 5 + 1)b_1) \bmod 2n$

$= (c + (5^n + 1)\beta n b) \bmod 2n$

$= c$

(because $5^n + 1$ is even)

We already proved that x_0 and $x_{2^{i-4}} = f_1^{2^{i-4}}(x_0) \bmod 2^i$ are in the same congruence class modulo 2^{i-4}.

$$\begin{aligned} x_{2\cdot 2^{i-4}} &= f_1^{2\cdot 2^{i-4}}(x_0) \bmod 2^i = f_1^{2^{i-4}}(f_1^{2^{i-4}}(x_0)) \bmod 2^i \\ &= f_1^{2^{i-4}}(x_{2^{i-4}}) \bmod 2^i \end{aligned}$$

$x_{2^{i-4}}$ and $x_{2.2^{i-4}}$ are in the same class. Thus x_0 and $x_{2.2^{i-4}}$ are in the same class.

Assume x_0 and $x_{k\cdot 2^{i-4}}$ are in the same class.

$$\begin{aligned} x_{(k+1)\cdot 2^{i-4}} &= f_1^{(k+1)\cdot 2^{i-4}}(x_0) \bmod 2^i = f_1^{2^{i-4}}(f_1^{k\cdot 2^{i-4}}(x_0)) \bmod 2^i \\ &= f_1^{2^{i-4}}(x_{k\cdot 2^{i-4}}) \bmod 2^i \end{aligned}$$

$x_{k\cdot 2^{i-4}}$ and $x_{(k+1)\cdot 2^{i-4}}$ are in the same class. Thus x_0 and $x_{(k+1)\cdot 2^{i-4}}$ are in the same class.

□

The vertices $x_0, x_{2^{i-4}}, x_{2\cdot 2^{i-4}}, x_{3\cdot 2^{i-4}}, \ldots, x_{15\cdot 2^{i-4}}$ are in the same congruence class modulo 2^{i-4} where $x_{k\cdot 2^{i-4}} = f_1^{k\cdot 2^{i-4}}(x_0) \bmod 2^i$. In other words, if $u, v \in V(G)$ are in the same congruence class modulo 2^{i-4}, then there exists an integer k, $0 < k < 16$, such that $v = f_1^{k\cdot 2^{i-4}}(u) \bmod 2^i$.

LEMMA 3.3.1 *Let $G(\{f_1, f_2\}, 2^i)$ be a one-dimensional linear congruential graph where f_1 and f_2 satisfy the conditions (1), (2), (3) and (4) in Theorem 3.3.2, and $i \geq 4$. If $u, v \in V(G)$ are in the same congruence class modulo 2^{i-4}, i.e., $u \bmod 2^{i-4} = v \bmod 2^{i-4}$, then*

$$f_2(u) \bmod 2^i = f_1^n(u) \bmod 2^i \iff f_2(v) \bmod 2^i = f_1^n(v) \bmod 2^i,$$

and therefore G is vertex-transitive with respect to a partition P of $V(G)$ where P consists of the congruence classes modulo 2^{i-4} on $V(G)$: $[0]_{2^{i-4}}, [1]_{2^{i-4}}, \cdots,$ $[2^{i-4} - 1]_{2^{i-4}}$.

Proof:

Let $v = u + m2^{i-4}$, $m \in N$, and assume $(u, u_1, u_2, \ldots, u_n)$ and $(v, v_1, v_2, \ldots, v_n)$ are part of the linear congruential sequence created by f_1 in G. Thus

$$\begin{aligned} u_1 &= f_1(u) \bmod 2^i = (5u + b_1) \bmod 2^i \\ u_2 &= f_1(u_1) \bmod 2^i = (5u_1 + b_1) \bmod 2^i \\ &\vdots \\ u_n &= f_1(u_{n-1}) \bmod 2^i = (5u_{n-1} + b_1) \bmod 2^i \\ &= f_2(u) \bmod 2^i = (9u + d_1) \bmod 2^i \end{aligned}$$

$$
\begin{aligned}
v_1 &= f_1(v) \bmod 2^i = (5v + b_1) \bmod 2^i \\
&= (5(u + m2^{i-4}) + b_1) \bmod 2^i = ((5u + b_1) + 5m2^{i-4}) \bmod 2^i \\
&= (u_1 + 5m2^{i-4}) \bmod 2^i \\
v_2 &= f_1(v_1) \bmod 2^i = (5(u_1 + 5m2^{i-4}) + b_1) \bmod 2^i \\
&= (u_2 + 5^2m2^{i-4}) \bmod 2^i \\
&\vdots \\
v_n &= (u_n + 5^nm2^{i-4}) \bmod 2^i = (9u + d_1 + 5^nm2^{i-4}) \bmod 2^i
\end{aligned}
$$

According to Theorem 3.3.4, $n = 2 + 4k_1 + 16k_2$ where $0 \leq k_1 \leq 3$, and $k_2 \in N$. Thus $5^n \bmod 16 = (5^2 * 5^{4(k_1+4k_2)}) \bmod 16 = (5^2 * 1) \bmod 16 = 9$, because $5^4 \bmod 16 = 1$.

Let $5^n = 16\alpha + 9$ where $\alpha \in N$,

$$
\begin{aligned}
v_n &= f_1^n(v) \bmod 2^i = (9u + d_1 + 5^nm2^{i-4}) \bmod 2^i \\
&= (9u + d_1 + (16\alpha + 9)m2^{i-4}) \bmod 2^i = (9u + d_1 + \alpha m2^i + 9m2^{i-4}) \bmod 2^i \\
&= (9(u + m2^{i-4}) + d_1) \bmod 2^i = (9v + d_1) \bmod 2^i \\
&= f_2(v) \bmod 2^i
\end{aligned}
$$

Now, we will prove that G is vertex-transitive with respect to the partition P. Assume vertices $u, v \in V(G)$ are in the same congruence class modulo 2^{i-4}. According to Theorem 3.3.5, there exists an integer k, $0 < k < 16$, such that $v = f_1^{k \cdot 2^{i-4}}(u) \bmod 2^i$.

Let $\alpha(x) = f_1^{k \cdot 2^{i-4}}(x) \bmod 2^i$ be a one-to-one mapping of $V(G)$ to $V(G)$. Thus $\alpha(u) = f_1^{k \cdot 2^{i-4}}(u) \bmod 2^i = v$.

Assume (x_0, x_1) is an edge in $E(G)$, then

Case 1: $x_1 = f_1(x_0) \bmod 2^i$

$$
\begin{aligned}
\alpha(x_0) &= f_1^{k \cdot 2^{i-4}}(x_0) \bmod 2^i \\
\alpha(x_1) &= \alpha(f_1(x_0) \bmod 2^i) = f_1^{k \cdot 2^{i-4}}(f_1(x_0) \bmod 2^i) \bmod 2^i \\
&= f_1(f_1^{k \cdot 2^{i-4}}(x_0) \bmod 2^i) \bmod 2^i = f_1(\alpha(x_0)) \bmod 2^i
\end{aligned}
$$

So $(\alpha(x_0), \alpha(x_1))$ is also an edge in $E(G)$.

Case 2: $x_1 = f_2(x_0) \bmod 2^i$

$\alpha(x_0) = f_1^{k\cdot 2^{i-4}}(x_0) \bmod 2^i$. Thus according to Theorem 3.3.5, x_0 and $\alpha(x_0)$ are in the same congruence class modulo 2^{i-4}. Assume $f_2(x_0) \bmod 2^i = f_1^{n'}(x_0) \bmod 2^i$, then $f_2(\alpha(x_0)) \bmod 2^i = f_1^{n'}(\alpha(x_0)) \bmod 2^i$.

$$\begin{aligned}
\alpha(x_1) &= \alpha(f_2(x_0) \bmod 2^i) = f_1^{k\cdot 2^{i-4}}(f_2(x_0) \bmod 2^i) \bmod 2^i \\
&= f_1^{k\cdot 2^{i-4}}(f_1^{n'}(x_0) \bmod 2^i) \bmod 2^i = f_1^{n'}(f_1^{k\cdot 2^{i-4}}(x_0) \bmod 2^i) \bmod 2^i \\
&= f_1^{n'}(\alpha(x_0)) \bmod 2^i = f_2(\alpha(x_0)) \bmod 2^i
\end{aligned}$$

So $(\alpha(x_0), \alpha(x_1))$ is also an edge in $E(G)$.

Thus α is an automorphism of G, i.e., for any two vertices u, v in the same congruence class modulo 2^{i-4}, there is an automorphism α such that $\alpha(u) = v$. G is vertex-transitive with respect to the partition P.

□

3.3.2 Graphs of Complex Form

Let $G(\{f\}, (2^i, s_2))$ be a two-dimensional linear congruential graph of complex form where $f(x,y) = (x,y)\begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, b_2)$, and $a_{11}, a_{21} \neq 0$. If (x_1, y_1), $(x_2, y_2) \in V(G)$, and (x_1, y_1) is adjacent to (x_2, y_2) via the generator f, then

$$\begin{aligned}
(x_2, y_2) &= f(x_1, y_1) \bmod (2^i, s_2) \\
&= (x_1, y_1)\begin{pmatrix} a_{11} & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, b_2) \bmod (2^i, s_2).
\end{aligned}$$

Therefore, $x_2 = (a_{11}x_1 + a_{21}y_1 + b_1) \bmod 2^i$, $y_2 = (y_1 + b_2) \bmod s_2$. Thus, y_2 depends only on y_1, but x_2 depends on both x_1 and y_1. For this reason, we cannot consider the graph G as a product of two one-dimensional graphs.

Although the graphs of complex form do not have the symmetric properties like the graphs of simple form described in previous subsection, they still have another symmetric property. We call it quarter-symmetry.

The reason why we call the property quarter symmetry is the following: if we consider the Hamiltonian cycle constructed by f_1 as a circle, the vertices $(x, y), ((x + 2^{i-2}) \bmod 2^i, y), ((x + 2 * 2^{i-2}) \bmod 2^i, y)$ and $((x + 3 * 2^{i-2}) \bmod 2^i, y)$ will divide the circle into four quarters, and for these four vertices, to apply f_2 to them is equivalent to apply f_1 n_{xy} times.

DEFINITION 3.3.3 Quarter-Symmetry

Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph in which f_1 generates a Hamiltonian cycle. We say the graph G is quarter-symmetric *if it has the following properties:*

- *For each vertex $(x, y) \in V(G)$, let $(x', y') = f_1^{k(2^{i-2} * s_2)}(x, y) \bmod (2^i, s_2)$ where $1 \leq k \leq 3$. Then $x' \bmod 2^{i-2} = x \bmod 2^{i-2}$, and $y' = y$.*
- *For each vertex $(x, y) \in V(G)$, there exists an n_{xy}, $1 < n_{xy} < (2^i * s_2)$, such that $f_2(x, y) \bmod (2^i, s_2) = f_1^{n_{xy}}(x, y) \bmod (2^i, s_2)$, furthermore,*

  $f_2((x + k2^{i-2}) \bmod 2^i, y) \bmod (2^i, s_2) = f_1^{n_{xy}}((x + k2^{i-2}) \bmod 2^i, y) \bmod (2^i, s_2)$ where $1 \leq k \leq 3$.

We illustrate the properties in Figure 3.1.

THEOREM 3.3.6 *Let $G(\{f\}, (2, s_2))$ be a two-dimensional linear congruential graph where $f(x, y) = (x, y)\begin{pmatrix} 5 & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, b_2)$, b_1 satisfies Lemma 3.2.1 (i.e., $\gcd(b_1, 2) = 1$), $\gcd(b_2, s_2) = 1$, and $s_2 = 4k + 1$, $k \in N$. Then, for every vertex $(x, y) \in V(G)$, $f^{s_2}(x, y) \bmod (2, s_2) = ((x + 1) \bmod 2, y)$.*

Proof:

Since y does not depend on x, we let $f_y(y) = y + b_2$. Because $\gcd(b_2, s_2) = 1$, the linear congruential sequence $y_0, y_1, \ldots, y_{s_2 - 1}$ defined by $y_j = f_y(y_{j-1}) \bmod s_2$, for $j \geq 1$, has a period of length s_2. Thus, for each $y \in \{0, 1, \ldots, s_2 - 1\}$, $f_y^{s_2}(y) \bmod s_2 = y$.

Assume $(x, y) \in V(G)$, and $f^{s_2}\ (x, y) \bmod (2, s_2) = (x', y)$. According to Lemma 3.2.3, f generates a Hamiltonian cycle of length $2 * s_2$ in G, i.e., $(x, y) = f^n(x, y) \bmod (2, s_2)$ only if $n = m(2 * s_2)$ where $m \in N$. Because $s_2 \neq m(2 * s_2)$, $x' \neq x$. In addition, since $0 \leq x, x' \leq 1$, $x' = (x + 1) \bmod 2$.

Thus, $f^{s_2}(x, y) \bmod (2, s_2) = ((x + 1) \bmod 2, y)$.

□

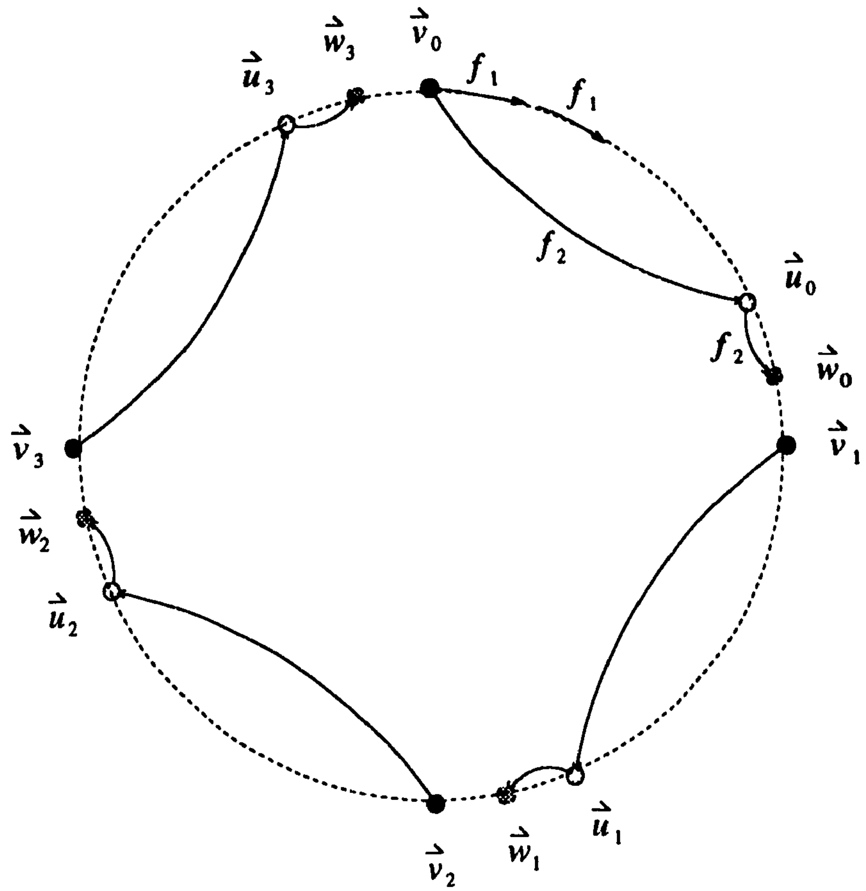


Figure 3.1: Two-dimensional Linear Congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ of complex form:

If $\vec{v_0} = (x_0, y_0)$, $\vec{v_1} = f_1^{\frac{1}{4}2^i s_2}(\vec{v_0}) \bmod (2^i, s_2)$, $\vec{v_2} = f_1^{\frac{2}{4}2^i s_2}(\vec{v_0}) \bmod (2^i, s_2)$, and $\vec{v_3} = f_1^{\frac{3}{4}2^i s_2}(\vec{v_0}) \bmod (2^i, s_2)$, then

$$\vec{v_1} = \begin{cases} ((x_0 + \frac{1}{4}2^i) \bmod 2^i, y_0) & \ldots \text{case A} \\ ((x_0 + \frac{3}{4}2^i) \bmod 2^i, y_0) & \ldots \text{case B} \end{cases}$$

$$\vec{v_2} = ((x_0 + \frac{2}{4}2^i) \bmod 2^i, y_0)$$

$$\vec{v_3} = \begin{cases} ((x_0 + \frac{3}{4}2^i) \bmod 2^i, y_0) & \ldots \text{case A} \\ ((x_0 + \frac{1}{4}2^i) \bmod 2^i, y_0) & \ldots \text{case B} \end{cases}$$

and if $\vec{u_0} = f_2(\vec{v_0}) \bmod (2^i, s_2) = f_1^n(\vec{v_0}) \bmod (2^i, s_2)$, $\vec{u_1} = f_2(\vec{v_1}) \bmod (2^i, s_2)$, $\vec{u_2} = f_2(\vec{v_2}) \bmod (2^i, s_2)$, and $\vec{u_3} = f_2(\vec{v_3}) \bmod (2^i, s_2)$, then

$$\vec{u_1} = f_1^n(\vec{v_1}) \bmod (2^i, s_2)$$
$$\vec{u_2} = f_1^n(\vec{v_2}) \bmod (2^i, s_2)$$
$$\vec{u_3} = f_1^n(\vec{v_3}) \bmod (2^i, s_2)$$

THEOREM 3.3.7 *Let $G(\{f\},(2^i,s_2))$ be a two-dimensional linear congruential graph where $f(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1,b_2)$, b_1 satisfies Lemma 3.2.1 (i.e., $\gcd(b_1,2^i) = 1$), $\gcd(b_2,s_2) = 1$, $s_2 = 4k+1$, $k \in N$. For each vertex $(x,y) \in V(G)$, if $f^n(x,y) \bmod (2^i,s_2) = (x',y')$, then $f^n(x,y) \bmod (2^{i+1},s_2)$ is equal to either (x',y') or $(x'+2^i,y')$.*

Proof:

Let $(x_0,y_0),(x_1,y_1),(x_2,y_2),\ldots,(x_{n-1},y_{n-1}),(x_n,y_n),\ldots$ be a linear congruential sequence created by f in the graph G, i.e., $(x_j,y_j) = f^j(x_0,y_0) \bmod (2^i,s_2)$ where $0 < j \leq n$. Assume there are m edge-changes between (x_0,y_0) and (x_n,y_n) ((x_n,y_n) is excluded), when G is extended. According to Lemma 3.2.6, if

1. m is even, then $f^n(x_0,y_0) \bmod (2^{i+1},s_2) = (x_n,y_n)$.

2. m is odd, then $f^n(x_0,y_0) \bmod (2^{i+1},s_2) = (x_n+2^i,y_n)$.

□

THEOREM 3.3.8 *Let $G(\{f\},(2^i,s_2))$ be a two-dimensional linear congruential graph as described in Theorem 3.3.7. If (x,y) is a vertex of G, then $f^{2^i*s_2}(x,y) \bmod (2^{i+1},s_2) = (x+2^i,y)$.*

Proof:

Let $G_2 = G(\{f\},(2^{i+1},s_2))$ be the extension of $G_1 = G(\{f\},(2^i,s_2))$. Since f generates a Hamiltonian cycle of period length $2^i * s_2$ in G_1, for every vertex $(x,y) \in V(G_1)$, $f^{2^i*s_2}(x,y) \bmod (2^i,s_2) = (x,y)$.

According to Theorem 3.3.7, $f^{2^i*s_2}(x,y) \bmod (2^{i+1},s_2)$ is equal to either (x,y) or $(x+2^i,y)$. Because f also generates a Hamiltonian cycle of period length $2^{i+1}*s_2$ in G_2, $f^{2^i*s_2}(x,y) \bmod (2^{i+1},s_2) \neq (x,y)$. Thus, $f^{2^i*s_2}(x,y) \bmod (2^{i+1},s_2) = (x+2^i,y)$.

□

THEOREM 3.3.9 *Let $G(\{f\},(2^i,s_2))$ be a two-dimensional linear congruential graph as described in Theorem 3.3.7, and $i \geq 2$. For each vertex $(x,y) \in V(G)$,*

$$\begin{aligned}
f^{\frac{1}{4}(2^i*s_2)}(x,y) \bmod (2^i, s_2) &= \begin{cases} ((x+\frac{1}{4}2^i) \bmod 2^i, y) & \ldots \mathit{case}\ A \\ ((x+\frac{3}{4}2^i) \bmod 2^i, y) & \ldots \mathit{case}\ B \end{cases} \\
f^{\frac{2}{4}(2^i*s_2)}(x,y) \bmod (2^i, s_2) &= ((x+\frac{2}{4}2^i) \bmod 2^i, y) \\
f^{\frac{3}{4}(2^i*s_2)}(x,y) \bmod (2^i, s_2) &= \begin{cases} ((x+\frac{3}{4}2^i) \bmod 2^i, y) & \ldots \mathit{case}\ A \\ ((x+\frac{1}{4}2^i) \bmod 2^i, y) & \ldots \mathit{case}\ B \end{cases}
\end{aligned}$$

Proof:

We prove the theorem by induction.

Basis: $i = 2$

Let $G_2 = G(\{f\},(4,s_2))$ be the extension of $G_1 = G(\{f\},(2,s_2))$.
For every vertex $(x,y) \in V(G_1)$, $f^{s_2}(x,y) \bmod (2,s_2) = ((x+1) \bmod 2, y)$ (Theorem 3.3.6), and thus $f^{s_2}(x,y) \bmod (4,s_2)$ is equal to either $((x+1) \bmod 4, y)$ or $((x+3) \bmod 4, y)$ (Theorem 3.3.7).

According to Theorem 3.3.8, for every vertex $(x,y) \in V(G_2)$, $f^{2*s_2}(x,y) \bmod (4,s_2) = ((x+2) \bmod 4, y)$, and if

1. $f^{s_2}(x,y) \bmod (4,s_2) = ((x+1) \bmod 4, y)$, then

   $f^{3*s_2}(x,y) \bmod (4,s_2) = ((x+3) \bmod 4, y)$.

2. $f^{s_2}(x,y) \bmod (4,s_2) = ((x+3) \bmod 4, y)$, then

   $f^{3*s_2}(x,y) \bmod (4,s_2) = ((x+1) \bmod 4, y)$.

Induction Step:

Let $G_2 = G(\{f\},(2^{i+1},s_2))$ be the extension of $G_1 = G(\{f\},(2^i,s_2))$. According to Theorem 3.3.8, for every vertex $(x,y) \in V(G_1)$,

$$f^{\frac{1}{2}(2^i*s_2)}(x,y) \bmod (2^i,s_2) = ((x+\frac{1}{2}2^i) \bmod 2^i, y).$$

We now consider the graph G_2. According to Theorem 3.3.7,

$$f^{\frac{1}{2}(2^i*s_2)}(x,y) \bmod (2^{i+1},s_2) = f^{\frac{1}{4}(2^{i+1}*s_2)}(x,y) \bmod (2^{i+1},s_2)$$

$$= \begin{cases} ((x+\tfrac{1}{2}2^i) \bmod 2^{i+1}, y) & = \quad ((x+\tfrac{1}{4}2^{i+1}) \bmod 2^{i+1}, y) \quad \ldots\text{case A} \\ ((x+\tfrac{1}{2}2^i+\tfrac{1}{2}2^{i+1}) \bmod 2^{i+1}, y) & = \quad ((x+\tfrac{3}{4}2^{i+1}) \bmod 2^{i+1}, y) \quad \ldots\text{case B} \end{cases}$$

According to Theorem 3.3.8, in case A,

$$\begin{aligned} f^{\frac{1}{4}(2^{i+1}*s_2)+\frac{1}{2}(2^{i+1}*s_2)}(x,y) \bmod (2^{i+1}, s_2) &= f^{\frac{3}{4}(2^{i+1}*s_2)}(x,y) \bmod (2^{i+1}, s_2) \\ &= ((x+\frac{1}{4}2^{i+1}+\frac{1}{2}2^{i+1}) \bmod 2^{i+1}, y) \\ &= ((x+\frac{3}{4}2^{i+1}) \bmod 2^{i+1}, y) \end{aligned}$$

and in case B,

$$\begin{aligned} f^{\frac{1}{4}(2^{i+1}*s_2)+\frac{1}{2}(2^{i+1}*s_2)}(x,y) \bmod (2^{i+1}, s_2) &= f^{\frac{3}{4}(2^{i+1}*s_2)}(x,y) \bmod (2^{i+1}, s_2) \\ &= ((x+\frac{3}{4}2^{i+1}+\frac{1}{2}2^{i+1}) \bmod 2^{i+1}, y) \\ &= ((x+\frac{1}{4}2^{i+1}) \bmod 2^{i+1}, y) \end{aligned}$$

Because $(x,y) \in V(G_2)$,

$$\begin{aligned} f^{\frac{1}{2}(2^{i+1}*s_2)}(x,y) \bmod (2^{i+1}, s_2) &= ((x+\frac{1}{2}2^{i+1}) \bmod 2^{i+1}, y) \\ &= ((x+\frac{2}{4}2^{i+1}) \bmod 2^{i+1}, y). \end{aligned}$$

□

LEMMA 3.3.2 *Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph where*

(1) ... $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, b_2)$,

(2) ... $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ c_{21} & 1 \end{pmatrix} + (d_1, d_2)$,

(3) ... b_1 satisfies Lemma 3.2.1, i.e., $\gcd(b_1, 2^i) = 1$,

(4) ... d_1 satisfies Lemma 3.2.2, i.e., $d_1 = 2\beta$, and $\gcd(\beta, 2^i) = 1$,

(5) ... $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, *and $s_2 = 4k+1$, $k \in N$.*

For each vertex $(x,y) \in V(G)$, if there exists an n_{xy} such that $f_2(x,y) \bmod (2^i, s_2) = f_1^{n_{xy}}(x,y) \bmod (2^i, s_2)$, *then*

$$f_2((x+\frac{m}{4}2^i) \bmod 2^i, y) \bmod (2^i, s_2) = f_1^{n_{xy}}((x+\frac{m}{4}2^i) \bmod 2^i, y) \bmod (2^i, s_2)$$

where $1 \le m \le 3$.

Proof:

To simplify the notation, we use n to denote n_{xy}.

Let (x_0, y_0) be a vertex of G. Since f_1 generates a Hamiltonian cycle in G, there exists an integer n such that

$$f_2(x_0, y_0) \bmod (2^i, s_2) = f_1^n(x_0, y_0) \bmod (2^i, s_2).$$

Assume $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}), (x_n, y_n)$ is a sub-sequence of the Hamiltonian cycle constructed by f_1. That is, $(x_j, y_j) = f_1(x_{j-1}, y_{j-1}) \bmod (2^i, s_2)$ where $0 < j \le n$.

$$\begin{aligned}
x_1 &= (5x_0 + a_{21}y_0 + b_1) \bmod 2^i \quad , \quad y_1 = (y_0 + b_2) \bmod s_2 \\
x_2 &= (5x_1 + a_{21}y_1 + b_1) \bmod 2^i \quad , \quad y_2 = (y_1 + b_2) \bmod s_2 \\
&\vdots \\
x_n &= (5x_{n-1} + a_{21}y_{n-1} + b_1) \bmod 2^i \quad , \quad y_n = (y_{n-1} + b_2) \bmod s_2
\end{aligned}$$

Because $(x_n, y_n) = f_2^n(x_0, y_0) \bmod (2^i, s_2)$,

$$x_n = (9x_0 + c_{21}y_0 + d_1) \bmod 2^i, y_n = (y_0 + d_2) \bmod s_2$$

Thus, $x_n = (5x_{n-1} + a_{21}y_{n-1} + b_1) \bmod 2^i = (9x_0 + c_{21}y_0 + d_1) \bmod 2^i$, and $y_n = (y_{n-1} + b_2) \bmod s_2 = (y_0 + d_2) \bmod s_2$.

Assume $(x'_0, y'_0), (x'_1, y'_1), \ldots, (x'_{n-1}, y'_{n-1}), (x'_n, y'_n)$ is also a sub-sequence of the Hamiltonian cycle constructed by f_1. That is, $(x'_j, y'_j) = f_1(x'_{j-1}, y'_{j-1}) \bmod (2^i, s_2)$, $0 < j \le n$. Let $x'_0 = (x_0 + \frac{m}{4}2^i) \bmod 2^i$, $1 \le m \le 3$, and $y'_0 = y_0$.
$y'_j = ((y'_{j-1} + b_2) \bmod s_2)$ does not depend on x'_{j-1}, and $y_j = (y_{j-1} + b_2) \bmod s_2$, so $y'_j = y_j$, for each j, $0 \le j \le n$.

$$\begin{aligned}
x'_1 &= (5x'_0 + a_{21}y'_0 + b_1) \bmod 2^i = (5(x_0 + \frac{m}{4}2^i) + a_{21}y_0 + b_1) \bmod 2^i \\
&= (\frac{5}{4}m2^i + (5x_0 + a_{21}y_0 + b_1)) \bmod 2^i = (\frac{m}{4}2^i + x_1) \bmod 2^i \\
x'_2 &= (5x'_1 + a_{21}y'_1 + b_1) \bmod 2^i = (5(x_1 + \frac{m}{4}2^i) + a_{21}y_1 + b_1) \bmod 2^i \\
&= (\frac{5}{4}m2^i + (5x_1 + a_{21}y_1 + b_1)) \bmod 2^i = (\frac{m}{4}2^i + x_2) \bmod 2^i
\end{aligned}$$

Induction Step:

Assume $x'_k = (\frac{m}{4}2^i + x_k) \bmod 2^i$, then

$$\begin{aligned} x'_{k+1} &= (5x'_k + a_{21}y'_k + b_1) \bmod 2^i = (5(\frac{m}{4}2^i + x_k) + a_{21}y_k + b_1) \bmod 2^i \\ &= (\frac{m}{4}2^i + (5x_k + a_{21}y_k + b_1)) \bmod 2^i = (\frac{m}{4}2^i + x_{k+1}) \bmod 2^i \end{aligned}$$

Thus, $f_1^n((x_0 + \frac{m}{4}2^i) \bmod 2^i, y_0) \bmod (2^i, s_2) = (x'_n, y'_n) = ((x_n + \frac{m}{4}2^i) \bmod 2^i, y_n)$

$$\begin{aligned} & f_2((x_0 + \frac{m}{4}2^i) \bmod 2^i, y_0) \bmod (2^i, s_2) \\ = \; & ((9(x_0 + \frac{m}{4}2^i) + c_{21}y_0 + d_1) \bmod 2^i, (y_0 + d_2) \bmod s_2) \\ = \; & ((\frac{9m}{4}2^i + (9x_0 + c_{21}y_0 + d_1)) \bmod 2^i, y_n) \\ = \; & ((\frac{m}{4}2^i + (9x_0 + c_{21}y_0 + d_1)) \bmod 2^i, y_n) \\ = \; & ((\frac{m}{4}2^i + x_n) \bmod 2^i, y_n) \end{aligned}$$

So $f_1^n((x_0 + \frac{m}{4}2^i) \bmod 2^i, y_0) \bmod (2^i, s_2) = f_2((x_0 + \frac{m}{4}2^i) \bmod 2^i, y_0) \bmod (2^i, s_2)$.

□

Chapter 4

Routing Algorithms of Two-Dimensional Linear Congruential Graphs

We will study the algorithms of global routing, distributed routing, finding disjoint paths and broadcasting for two-dimensional linear congruential graphs. The algorithms that we will present are based on the symmetric properties of graphs discussed in the previous chapter.

In the description of the algorithms, for a vertex $\vec{u}$ of $G(\{f_1, f_2\}, (2^i, s_2))$, we will denote an edge e from $\vec{u}$ to $\vec{v}$ as 0 if e is generated by f_1 on $\vec{u}$, 1 if e is generated by f_2 on $\vec{u}$, 2 if e is generated by f_1 on $\vec{v}$, or 3 if e is generated by f_2 on $\vec{v}$. A path from a source vertex $\vec{u}$ can be represented by a string on an alphabet $\{0, 1, 2, 3\}$ of four letters, and it is considered as a sequence of generators applied sequentially on the vertices that the path traverses along, rather than the sequence of vertices.

For example, a path $\vec{u} \rightarrow f_1(\vec{u}) \bmod (2^i, s_2) \rightarrow f_2(f_1(\vec{u})) \bmod (2^i, s_2) \rightarrow f_1^{-1}(f_2(f_1(\vec{u}))) \bmod (2^i, s_2) \rightarrow f_2^{-1}(f_1^{-1}(f_2(f_1(\vec{u})))) \bmod (2^i, s_2)$ can be represented by the string "0123". We can use two bits to represent an edge, and thus this path is represented by eight bits. For a graph of diameter less than 16, the shortest paths between any two vertices can therefore be represented by four bytes.

4.1 Global Routing

To determine the shortest paths between two vertices of a hypercube and a de Bruijn graph is very straightforward. For a hypercube, it can be done by a series of changing one bit of a string. For a de Bruijn graph, it can be done by a series of right- or left-shifts of a string. The definition of adjacencies in linear congruential graphs is more complex than those in hypercubes and de Bruijn graphs. Thus we do not have yet a simple and fast mathematical transformations on the coordinates of the vertices to determine the shortest path between two vertices of a linear congruential graph. Our global routing algorithm is a method of searching the destination vertex along all the possible edges in a recursive way. The algorithm is similar to the *Dijkstra's* algorithm, and it can be introduced as follows.

Given a source vertex $\vec{u}$ and a destination vertex $\vec{v}$ of a two-dimensional linear congruential graph of degree 4, if $\vec{u}$ is not equal to $\vec{v}$, in step 1, we search $\vec{v}$ along all of the four edges incident with $\vec{u}$. If one of the neighbors of $\vec{u}$ is $\vec{v}$, the shortest path is the edge incident with both $\vec{u}$ and $\vec{v}$. Otherwise the searching process has to continue and goes to step 2.

These four edges incident with $\vec{u}$ are called the *searching paths* p_j of length one where $1 \leq j \leq 4$, and they may be the head of the shortest paths. The *destinations* $\vec{u}_j$ of the searching paths p_j are marked "*visited*". The edges that are incident with $\vec{u}_j$ but not the last edge of p_j are called the *outgoing edges* of $\vec{u}_j$.

In step 2, each searching path p_j is extended along three outgoing edges of $\vec{u}_j$, separately. If a vertex $\vec{u}_{jk}$ adjacent to $\vec{u}_j$ via an outgoing edge of $\vec{u}_j$ is not marked "visited", a new searching path p_{jk} of length 2 is created by appending the outgoing edge to p_j, and $\vec{u}_{jk}$ is marked "visited". If $\vec{u}_{jk}$ is equal to $\vec{v}$, p_{jk} is the shortest path from $\vec{u}$ to $\vec{v}$.

After step 2, if none of the searching paths p_{jk} of length 2 is a shortest path from $\vec{u}$ to $\vec{v}$, searching has to continue in a similar way until the destination vertex $\vec{v}$ is found.

The searching process is quite time-consuming. It is not efficient to execute the process whenever a vertex $\vec{u}$ needs to send a message to a given vertex $\vec{v}$. Instead, we can execute it only once by creating a table that consists of the shortest paths from $\vec{u}$ to each of other vertices of the graph. We will call the table ***shortest-path table*** for the vertex $\vec{u}$. For a vertex of $G(\{f_1, f_2\}, (2^i, s_2))$, the shortest-path table contains $2^i \times s_2$ entries, and each entry may contain more than one shortest path. There are $2^i \times s_2$ vertices, therefore the graph needs $2^i \times s_2$ shortest-path tables for global routing. The number of shortest-path tables required increases with the size of the graph.

In the following subsections, we will introduce the algorithm for constructing the shortest-path tables and then discuss how to reduce the number of tables based on the symmetric properties of graphs.

4.1.1 Global Routing in Graphs of Simple Form

Before describing the algorithm for constructing the shortest-path tables for a graph $G(\{f_1, f_2\}, (2^i, s_2))$, we first introduce the data structures that are used in the algorithm.

In a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ of simple form, $G = G(\{f_{1_x}, f_{2_x}\}, 2^i) \circ G(\{f_{1_y}, f_{2_y}\}, s_2)$, since the value of the coordinate x is independent of the value of the coordinate y, we can construct two tables: *x-edge table* and *y-edge table*. The x-edge table has 2^i rows indexed from 0 to $2^i - 1$ and 4 columns indexed from 0 to 3. An entry in row u_x and column e represents the vertex v_x of $G(\{f_{1_x}, f_{2_x}\}, 2^i)$ that is adjacent to the vertex u_x via the edge e. That is, if $e = 0$, $v_x = f_{1_x}(u_x) \bmod 2^i$. If $e = 1$, $v_x = f_{2_x}(u_x) \bmod 2^i$. If $e = 2$, $u_x = f_{1_x}(v_x) \bmod 2^i$. If $e = 3$, $u_x = f_{2_x}(v_x) \bmod 2^i$.

Similarly, the y-edge table has s_2 rows indexed from 0 to $s_2 - 1$ and 4 columns indexed from 0 to 3. An entry in row u_y and column e represents the vertex v_y of $G(\{f_{1_y}, f_{2_y}\}, s_2)$ that is adjacent to the vertex u_y via the edge e. That is, if $e = 0$, $v_y = f_{\cdot_y}(u_y) \bmod s_2$. If $e = 1$, $v_y = f_{2_y}(u_y) \bmod s_2$. If $e = 2$, $u_y = f_{1_y}(v_y) \bmod s_2$. If $e = 3$, $u_y = f_{2_y}(v_y) \bmod s_2$.

A *searching-path table* for a given source vertex (s_x, s_y) has s_2 rows indexed from 0 to $s_2 - 1$ and D columns indexed from 1 to D where D is the diameter of the graph G. Each entry is a pointer to a list of *searching path record.* A searching path record consists of a searching path p, the coordinate x, d_x, of the destination of the searching path p, and a pointer "*next*" to the next searching path record.

In the searching-path table for a source vertex (s_x, s_y), a searching path record $(p, d_x, next)$ in the list pointed by the entry in row d_y and column l means that the searching path p is a (shortest) path of length l from (s_x, s_y) to (d_x, d_y).

A *shortest-path table* for a given source vertex (s_x, s_y) has 2^i rows indexed from 0

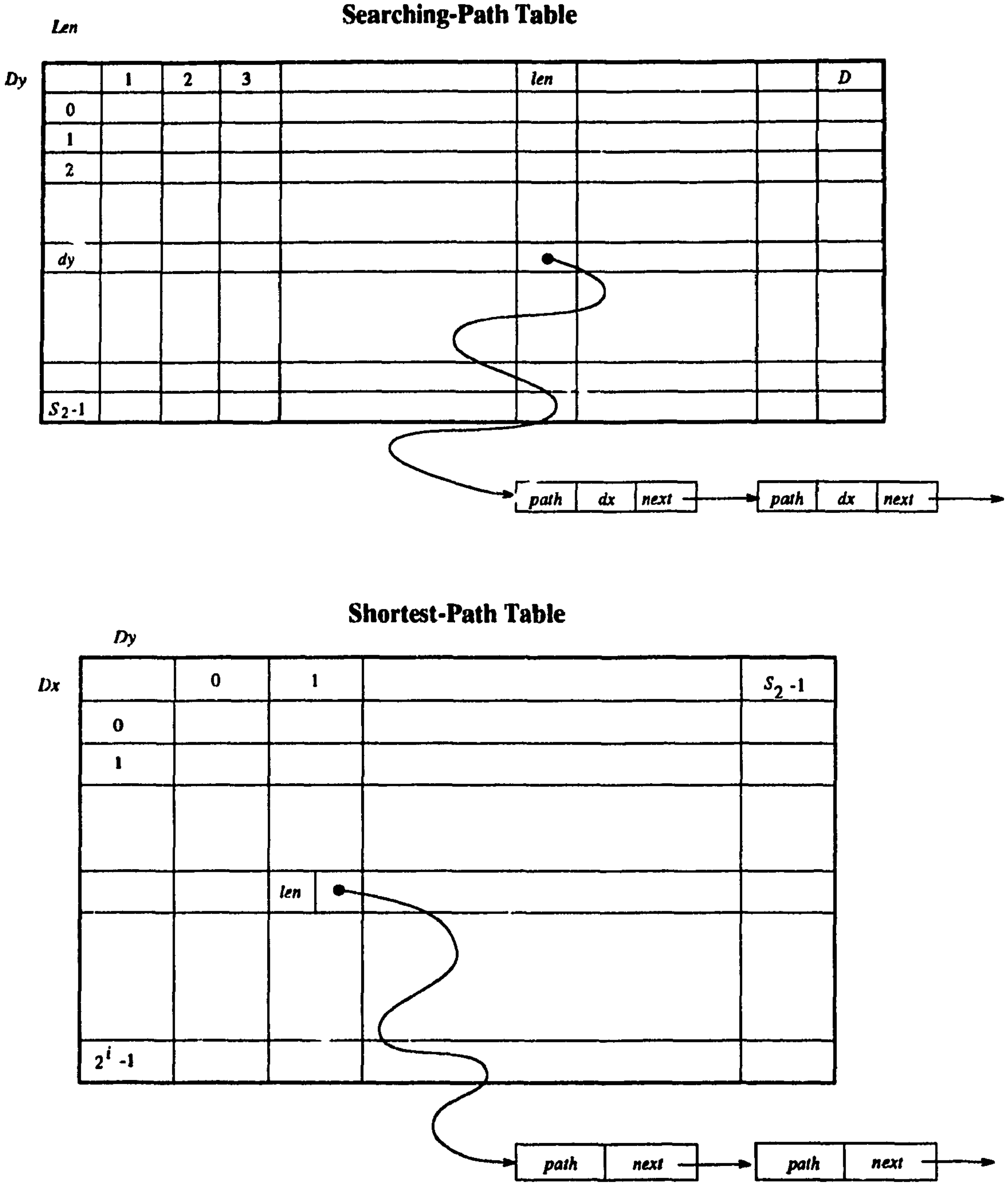


Figure 4.1: Data structure of global routing

to $2^i - 1$ and s_2 columns indexed form 0 to $s_2 - 1$. Each entry is a record consisting of the length l of the shortest paths and a pointer *pathpt* to a list of *shortest path records.* A shortest path record consists of a path p and a pointer ”*next*” to the next shortest path record. Figure 4.1 illustrates the searching-path and shortest-path tables.

In the shortest-path table for a source vertex (s_x, s_y), a shortest path record $(p, next)$ in the list pointed by the entry $(l, pathpt)$ in row d_y and column d_x means that p is one of the shortest paths from (s_x, s_y) to (d_x, d_y) and its length is l.

Our algorithm can be described in pseudo-code as follows.

Algorithm: construction of shortest-path table;

- input :

  f_1, f_2: the generators of a two-dimensional linear congruential graph G of simple form.

  $(2^i, s_2)$: the size of the graph G.

  (s_x, s_y): a source vertex of G.

- output : the shortest-path table for (s_x, s_y).

```
begin
  construct the x_edge table;
  construct the y_edge table;
  len := 1;
  for e := 0 to 3 do               {* for each edge incident with     }
  begin                            {  (sx, sy)                       *}
    dx := x_edge_table[sx, e];
    dy := y_edge_table[sy, e];
    new(searching_path_rec);       {* create a searching path record *}
    searching_path_rec^.dx := dx;
    searching_path_rec^.path := e;
                                   {* put it in searching_path table *}
    searching_path_rec^.next := searching_path_table[dy, len];
    searching_path_table[dy, len] := searching_path_rec;

    new(shortest_path_rec);        {* create a shortest path record  *}
    shortest_path_rec^.path := e;
                                   {* put it in shortest_path table  *}
    shortest_path_rec^.next := nil;
    shortest_path_table[dx, dy].len := len;
  shortest_path_table[dx, dy].pathpt := shortest_path_rec;
end; {of the for loop}

all_found := false;
while not all_found do
  for y := 0 to s2 - 1 do      {* for each entry of length "len"  }
  begin                        {  in the searching_path table    *}

    current := searching_path_table[y, len];
    while current <> nil do    {* for each searching path record *}
    begin

      for e := 0 to 3 do       {* for each edge incident with the }
      begin                    {  destination of the searching    }
                               {  path                           *}
        if e <> inverse(current^.path mod 4) then
        begin                  {* if e is an outgoing edge       *}

          dx := x_edge_table[current^.dx, e];
          dy := y_edge_table[y, e];
          if (shortest_path_table[dx, dy].len = 0) or
             (shortest_path_table[dx, dy].len = len + 1) then
          begin                {* no shortest path has been found,}
                               {  or this path is one of the      }
                               {  shortest paths                 *}
            new{searching_path_rec};
                               {* create a searching path record  }
                               {  of length (len + 1)            *}
            searching_path_rec^.path := current^.path * 4 + e;
            searching_path_rec^.dx := dx;
                               {* put it in searching_path table *}
            searching_path_rec^.next :=
                            searching_path_table[dy, len+1];
            searching_path_table[dy, len+1] := searching_path_rec;

            new{shortest_path_rec};
                               {* create a shortest path record  *}
            shortest_path_rec^.path := current^.path * 4 + e;
            shortest_path_rec^.next :=
                            shortest_path_table[dx, dy].pathpt;
                               {* put it in shortest_path table  *}
            shortest_path_table[dx, dy].len := len + 1;
```

```
              shortest_path_table[dx, dy].pathpt:=searching_path_rec;
            end; {of if}
          end; {if e <> inv..}
        end; {of for e}
        current := current^.next;
                                         {* process the next searching path*}
      end; {of while current}
    end; {of for y}
    if every entry (except (sx, sy)) of the shortest_path table has
      at least one shortest path
    then
      all_found := true;
  end; {of  while not all_found}
end.
```

As we mentioned, for a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 2^i) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 2^i) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k+1$, the number of distinct shortest-path tables can be reduced to 2^{i-4}.

Let $G_x = G(\{f_{1_x}, f_{2_x}\}, 2^i)$, and $G_y = G(\{f_{1_y}, f_{2_y}\}, s_2)$ where $f_{1_x}(x) = 5x + b_1$, $f_{2_x}(x) = 9x + d_1$, $f_{1_y}(y) = y + b_2$, $f_{2_y}(y) = y + d_2$. Then $G = G_x \circ G_y$. Since G_x is vertex-transitive with respect to a partition P where P consists of the congruence classes modulo 2^{i-4}, and G_y is vertex-transitive, for a given vertex (s_x, s_y) of G, there exists an automorphism $\alpha = (\alpha_x, \alpha_y)$ such that $\alpha_x(s_x) = s'_x$ where $s'_x = s_x \bmod 2^{i-4}$, $\alpha_y(s_y) = s'_y = 0$.

If we consider a path as a sequence of generators applied sequentially on the vertices that the path traverses along, rather than the sequence of vertices, then given a path $p = (e_1, e_2, \ldots, e_l)$, $0 \leq e_i \leq 3$, from (s_x, s_y) to (d_x, d_y), p is also a path from (s'_x, s'_y) to (d'_x, d'_y) where $d'_x = \alpha_x(d_x)$, $d'_y = \alpha_y(d_y)$. Consequently, we only have to construct 2^{i-4} shortest-path tables for source vertices $(0,0), (1,0), \ldots, (2^{i-4}-1, 0)$.

Let (s_x, s_y) be a source vertex and (d_x, d_y) be a destination vertex. One of the methods to find the shortest paths between (s_x, s_y) and (d_x, d_y) is to find the corresponding automorphism $\alpha = (\alpha_x, \alpha_y)$ such that $\alpha_x(s_x) = s_x \bmod 2^{i-4}$, $\alpha_y(s_y) = 0$, and then to get the paths from the corresponding entry of $(\alpha_x(d_x), \alpha_y(d_y))$ in the shortest-path table for the source vertex ($s_x \bmod 2^{i-4}, 0$).

The simplest way to determine α_x and α_y is to assume $\alpha_x(x) = f_{1_x}^{n_x}(x)$, $\alpha_y(y) = f_{1_y}^{n_y}(y)$ such that $f_{1_x}^{n_x}(s_x) = s_x \bmod 2^{i-4}$, $f_{1_y}^{n_x}(s_y) = 0$. The average times of iterative calculation to find n_x and n_y is 2^{i-1} and $\frac{s_2}{2}$, respectively. This method is time-consuming, and thus another table-checking method is proposed as follows.

This method is based on the structure of the Hamiltonian cycles generated by f_{1_x} and f_{1_y}. We first create an *x-Hamiltonian array* and a *y-Hamiltonian array.* The x-Hamiltonian array has 2^i entries indexed from 0 to $2^i - 1$. The content of the first entry is (vertex) 0, and the content of the jth entry is $f_{1_x}^j(0) \bmod 2^i$. Similarly, the y-Hamiltonian array has s_2 entries indexed from 0 to $s_2 - 1$. The content of the first entry is (vertex) 0, and the content of the jth entry is $f_{1_y}^j(0) \bmod s_2$.

We then create an *x-position array* and a *y-position array* according to the x-Hamiltonian array and y-Hamiltonian array. The x-position array has 2^i entries indexed from 0 to $2^i - 1$. The content of the first entry is 0, and the content of the xth entry is j where $f_{1_x}^j(0) \bmod 2^i = x$. That is, the vertex x is in the position j on the Hamiltonian cycle generated by f_{1_x}. Similarly, the y-position array has s_2 entries indexed from 0 to $s_2 - 1$. The content of the first entry is 0, and the content of the yth entry is j where $f_{1_y}^j(0) \bmod s_2 = y$. That is, the vertex y is in the position j on the Hamiltonian cycle generated by f_{1_y}.

Thus, whenever we want to find the shortest paths from a vertex (s_x, s_y) to a vertex (d_x, d_y), we just use the following formulas to get (s'_x, s'_y) and (d'_x, d'_y),

$$\begin{aligned} s'_x &= s_x \bmod 2^{i-4} \\ s'_y &= 0 \\ d'_x &= x\_Hamiltonian[(x\_position[s'_x] + x\_position[d_x] - x\_position[s_x]) \bmod 2^i] \\ d'_y &= y\_Hamiltonian[(y\_position[d_y] - y\_position[s_y]) \bmod s_2] \end{aligned}$$

and then check the corresponding entry in the shortest-path table for source vertex $(s'_x, 0)$ to find the paths from (s_x, s_y) to (d_x, d_y).

4.1.2 Global Routing in Graphs of Complex Form

The algorithms for constructing the shortest-path tables in a two-dimensional linear congruential graph of complex form is similar to what we described in the previous subsection, but the x-edge table here is generated with a different structure.

In a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ of complex form, if a vertex (x, y) is adjacent to a vertex (x', y'), x' is dependent on both x and y. Therefore, the x-edge table should be modified to have $2^i \times s_2$ rows. The jth row corresponds to the vertex $(x, y) = (j \text{ div } s_2,\ j \bmod s_2)$, and the content of the entry in jth row and eth column is x' where $(x', y') = f_1(x, y) \bmod (2^i, s_2)$ if $e = 0$, $(x', y') = f_2(x, y) \bmod (2^i, s_2)$ if $e = 1$, $(x, y) = f_1(x', y') \bmod (2^i, s_2)$ if $e = 2$, and $(x, y) = f_2(x', y') \bmod (2^i, s_2)$ if $e = 3$.

In addition, for a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ of complex form where $f_1(x, y) = (x, y) \begin{pmatrix} 5 & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x, y) = (x, y) \begin{pmatrix} 9 & 0 \\ c_{21} & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 2^i) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 2^i) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k + 1$, $k \in N$, since the graph G is quarter-symmetric, the number of distinct shortest-path tables required is $2^{i-2} \times s_2$.

To find the shortest paths from (s_x, s_y) to (d_x, d_y), we first create a *Hamiltonian array* and a *position table* in the similar way described in the previous subsection. Each entry of the Hamiltonian array is a record (x, y). Then we use the following formulas to get (s'_x, s'_y), (d'_x, d'_y),

$$s'_x = s_x \bmod 2^{i-2}$$
$$s'_y = s_y$$
$$d'_x = Hamiltonian[(position[s'_x, s'_y] + position[d_x, d_y] - position[s_x, s_y]) \bmod 2^i \times s_2].x$$
$$d'_y = d_y$$

and check the corresponding entry in the shortest-path table for (s'_x, s'_y) to find the paths from (s_x, s_y) to (d_x, d_y).

4.1.3 Comparison of the Global Routing in Graphs of Simple Form and Complex Form

The structure of graphs belonging to the family of graphs of complex form ($\mathcal{G}_c$) is more complex than the structure of graphs belonging to the family of graphs of simple form ($\mathcal{G}_s$). However, the diameters of graphs in $\mathcal{G}_c$ are smaller than those of graphs in $\mathcal{G}_s$ in general. For example, let $G_s = G(\{f_1, f_2\}, (2^{13}, 9))$ be a graph in

y x	0	1	2	3	4	5	6	7	8
0		33	122 212 221	322 232 223	0000	2222	100 010 001	300 030 003	11
1	323 233 332	1222 2122 2212 2221	3222 2322 2232 2223	000	0111 1011 1101 1110	10 01	30 03	112 121 211	2
2	30000, 03000, 00300, 00030, 00003, 12222, 21222, 22122, 22212, 22221,	0011 1001 0101 1010 0110 1100	00	111	1	3	333	22	2323 3223 2233 3322 2332 3232
3	110 011 101	0	033 303 330	21 12	23 32	2333 3233 3323 3332	222	1000 0100 0010 0001	3000 0300 0030 0003

Table 4.1: The shortest-path table for vertex $(0,0)$ of G_s

$\mathcal{G}_s$ where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2,4)$, and $G_c = G(\{f_1', f_2'\}, (2^{13}, 9))$ be a graph in $\mathcal{G}_c$ where $f_1'(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 2 & 1 \end{pmatrix} + (3,1)$, $f_2'(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 2 & 1 \end{pmatrix} + (2,4)$. The maximum distance from vertex $(0,0)$ to any vertex is 14 for G_s and 12 for G_c. Table 4.1 and 4.2 give examples of shortest-path table for a vertex of a graph of simple form and for a vertex of a graph of complex form, respectively. The shortest-path tables are created for vertex $(0,0)$ of $G_s = G(\{f_1, f_2\}, (4,9))$ and $G_c = G(\{f_1', f_2'\}, (4,9))$.

Assume the process of constructing a shortest-path table for a graph $G(\{f_1, f_2\}, (2^i, s_2))$ will take t units of time. If the graph G is of simple form, 2^{i-4} distinct shortest-path tables are required, and thus it takes $2^{i-4} \times t$ units of time to construct these tables. If the graph G is of complex form, $2^{i-2} \times s_2$ distinct shortest-path tables are required, and thus it takes $2^{i-2} \times s_2 \times t$ units of time to construct these tables.

In the next section, we will introduce another global routing algorithm for the graphs of simple form. It only needs to create a unique shortest-path table.

x \ y	0	1	2	3	4	5	6	7	8
0		33	00	223 322	012 210	3	010	22	11
1	101 233 332	103	330	12	32	10	30 03	121	2
2	1230 3210 2301 2103 0123 1032	0101, 2303, 3032, 0332, 3230, 0110 1100	122	111 232	1	230	100 001 333	1232 3212 2321 1111 1223	1210, 2101, 0121, 3223, 2233, 3322, 2332, 1012
3	110 011 323	0	033 303	21	23	01	2300 3301 1033	112 211	321 123

Table 4.2: The shortest-path table for vertex (0,0) of G_r

4.2 Global Routing in Graphs of Simple Form with a Unique Path Table

As described in the previous section, the number of distinct shortest-path tables required for the global routing algorithm increases with the size of the graph. To generate these tables for a graph is time consuming especially when the size of the graph is large. In order to speed up the global routing, the research in this section focuses on reducing the number of distinct shortest-path tables required. An efficient algorithm is proposed for graphs of simple form. This algorithm needs to generate only one modified shortest-path table along with a size-independent mapping table.

Let $G(\{f_1, f_2\}, (16, s_2))$ be a two-dimensional linear congruential graph of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 16) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 16) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k + 1$. Then $G = G_x \circ G_y$, and G_x, G_y are vertex-transitive.

Let (s_x, s_y) and (d_x, d_y) be two vertices of G. If p is a path from (s_x, s_y) to (d_x, d_y), then p is also a path from $(0,0)$ to $(\alpha_x(d_x), \alpha_y(d_y))$ where $\alpha = (\alpha_x, \alpha_y)$ is an automorphism on $V(G)$ such that $\alpha_x(s_x) = 0$, and $\alpha_y(s_y) = 0$. If there is a shortest-path table for the source vertex $(0, 0)$, then given any two vertices of G, we can find the shortest paths between them by checking this table.

Let $G' = G(\{f_1, f_2\}, (32, s_2))$ be the extension of $G(\{f_1, f_2\}, (16, s_2))$, and (s_x, s_y), (d_x, d_y) be two vertices of G'. If p is a path from (s_x, s_y) to (d_x, d_y), then according to Lemma 3.2.6 (symmetry of extension), p is also a path either from $(0, 0)$ to $(\alpha_x(d_x \bmod 16), \alpha_y(d_y))$ or from $(0,0)$ to $(\alpha_x(d_x \bmod 16) + 16, \alpha_y(d_y))$ where $\alpha = (\alpha_x, \alpha_y)$ is an automorphism on $V(G)$ such that $\alpha_x(s_x \bmod 16) = 0$, and $\alpha_y(s_y) = 0$.

On the contrary, if p_1 and p_2 are paths from $(0,0)$ to $(\alpha_x(d_x \bmod 16), \alpha_y(d_y))$ and to $(\alpha_x(d_x \bmod 16) + 16, \alpha_y(d_y))$, respectively, one of the following cases may occur.

1. Both p_1 and p_2 are paths from (s_x, s_y) to (d_x, d_y).

2. One of them is a path from (s_x, s_y) to (d_x, d_y), and another is a path from (s_x, s_y) to $((d_x + 16) \bmod 32, d_y)$.

3. Both p_1 and p_2 are paths from (s_x, s_y) to $((d_x + 16) \bmod 32, d_y)$.

Even though there is a shortest-path table for the source vertex $(0, 0)$, we probably cannot find a path from (s_x, s_y) to (d_x, d_y) by checking these two entries corresponding to $(\alpha_x(d_x \bmod 16), \alpha_y(d_y))$ and $(\alpha_x(d_x \bmod 16) + 16, \alpha_y(d_y))$ in the table. It is also likely that some of the shortest paths in an entry are paths from (s_x, s_y) to (d_x, d_y), and the others in the same entry are paths from (s_x, s_y) to $((d_x + 16) \bmod 32, d_y)$. Thus, every path in the corresponding entries has to be checked whether or not it is a path from (s_x, s_y) to (d_x, d_y). The number of corresponding entries (2^{i-4}) increases with the size $2^i \times s_2$ of the graph. The checking procedure is time consuming especially when the size of the graph is large.

To overcome the above disadvantages, the shortest-path table is modified such that given any two paths in the table, when they are applied on the same source vertex,

1. their destinations are the same, if they are in the same entry.

2. their destinations are different, if they are in different entries.

We will call this modified table a *one-to-one mapping path table*. The algorithm for constructing the modified table is introduced in the following subsection, and the related proofs are also given.

4.2.1 Construction of the One-to-one Mapping Path Table

Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} a_{11} & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, and $f_2(x,y) = (x,y)\begin{pmatrix} c_{11} & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, Let $f_{1_x}(x) = a_{11}x + b_1$, $f_{1_y}(y) = y + b_2$, $f_{2_x}(x) = c_{11}x + d_1$, and $f_{2_y}(y) = y + d_2$. Let $\bar{a}_{11}$, $\bar{b}_1$, $\bar{b}_2$, $\bar{c}_{11}$, $\bar{d}_1$ and $\bar{d}_2$ be constants such that $f_{1_x}^{-1}(x) = \bar{a}_{11}x + \bar{b}_1$, $f_{1_y}^{-1}(y) = y + \bar{b}_2$, $f_{2_x}^{-1}(x) = \bar{c}_{11}x + \bar{d}_1$, and $f_{2_y}^{-1}(y) = y + \bar{d}_2$. Let p be a path in G from (s_x, s_y) to (d_x, d_y), and n_1, n_2, n_3, n_4 be the numbers of generators f_1, f_2, f_1^{-1}, f_2^{-1} in p, respectively. Then

$$\begin{aligned} d_x &= ((a_{11}^{n_1} \cdot c_{11}^{n_2} \cdot \bar{a}_{11}^{n_3} \cdot \bar{c}_{11}^{n_4}) \times s_x + R_x(p)) \bmod 2^i \\ d_y &= (s_y + R_y(p)) \bmod s_2 \end{aligned}$$

where R_x consists of the items that do not contain s_x, and R_y consists of the items that do not contain s_y. If we define $A(p)$ to be $(a_{11}^{n_1} \cdot c_{11}^{n_2} \cdot \bar{a}_{11}^{n_3} \cdot \bar{c}_{11}^{n_4})$, then $d_x = A(p) \times s_x + R_x(p)$.

THEOREM 4.2.1 *Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} a_{11} & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, and $f_2(x,y) = (x,y)\begin{pmatrix} c_{11} & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$. Let p_1 be a path in G from $(0,0)$ to (d_{x0}, d_{y0}) and also a path from (s_x, s_y) to (d_x, d_y). Let p_2 be a path in G from $(0,0)$ to (d'_{x0}, d'_{y0}) and also a path from (s_x, s_y) to (d'_x, d'_y), and $A(p_1) = A(p_2)$. If $d_{x0} = d'_{x0}$, then $d_x = d'_x$, else $d_x \neq d'_x$. If $d_{y0} = d'_{y0}$, then $d_y = d'_y$, else $d_y \neq d'_y$.*

Proof:

$$\begin{aligned} d_{x0} &= (A(p_1) \times 0 + R_x(p_1)) \bmod 2^i. \text{ Thus } R_x(p_1) \bmod 2^i = d_{x0}. \\ d_x &= (A(p_1) \times s_x + R_x(p_1)) \bmod 2^i = (A(p_1) \times s_x + d_{x0}) \bmod 2^i \\ d'_{x0} &= (A(p_2) \times 0 + R_x(p_2)) \bmod 2^i. \text{ Thus } R_x(p_2) \bmod 2^i = d'_{x0}. \\ d'_x &= (A(p_2) \times s_x + R_x(p_2)) \bmod 2^i = (A(p_2) \times s_x + d'_{x0}) \bmod 2^i \end{aligned}$$

Since $A(p_1) = A(p_2)$, if $d_{x0} = d'_{x0}$, then $d_x = d'_x$, else $d_x \neq d'_x$.

$$\begin{aligned}
d_{y0} &= (0 + R_y(p_1)) \bmod s_2. \text{ Thus } R_y(p_1) \bmod s_2 = d_{y0}. \\
d_y &= (s_y + R_y(p_1)) \bmod s_2 = (s_y + d_{y0}) \bmod s_2 \\
d'_{y0} &= (0 + R_y(p_2)) \bmod s_2. \text{ Thus } R_y(p_2) \bmod s_2 = d'_{y0}. \\
d'_y &= (s_y + R_y(p_2)) \bmod s_2 = (s_y + d'_{y0}) \bmod s_2
\end{aligned}$$

If $d_{y0} = d'_{y0}$, then $d_y = d'_y$, else $d_y \neq d'_y$.

□

Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 2^i) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 2^i) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k+1$, $k \in N$. According to the vertex transitivity of $G' = G(\{f_1, f_2\}, (16, s_2))$ and the property of edge-change in the graph G, a path in G from a vertex (s_x, s_y) to a vertex in $\{(d_x, d_y), (d_x+16, d_y), \ldots, (d_x+16\times(2^{i-4}-1), d_y)\}$ could be found only in the entries corresponding to the vertices in $\{(\alpha_x(d_x), \alpha_y(d_y)), (\alpha_x(d_x)+16, \alpha_y(d_y)), \ldots, (\alpha_x(d_x)+16\times(2^{i-4}-1), \alpha_y(d_y))\}$ of the shortest-path table for the source vertex $(0,0)$ of G where $0 \leq d_x \leq 15$, and $\alpha = (\alpha_x, \alpha_y)$ is an automorphism on $V(G')$ such that $\alpha_x(s_x \bmod 16) = 0$, $\alpha_y(s_y) = 0$. Thus if all paths in these entries have the same $A(p)$, according to the above theorem, there is a one-to-one mapping between $\{(d_x+16\times j, d_y) \mid 0 \leq j \leq 2^{i-4}-1\}$ and $\{(\alpha_x(d_x)+16\times k, \alpha_y(d_y)) \mid 0 \leq k \leq 2^{i-4}-1\}$.

Let $R = \{(x_1, y_1), (x_2, y_2) \mid x_1 \bmod 16 = x_2 \bmod 16, \text{ and } y_1 = y_2\}$ be a relation on $V(G)$. Thus R is a equivalence relation. There are $16 \times s_2$ equivalence classes. Each of them consists of 2^{i-4} vertices. In the shortest-path table for $(0,0)$ of G, if all paths in the entries corresponding to the vertices that are in the same equivalence class have the same $A(p)$, the table is called a *one-to-one mapping path table.*

The following is an algorithm for constructing a one-to-one mapping path table. In the algorithm, we use Ap to denote $A(p)$, and there are two data structures *Ap_list* and *Ap_counter*. *Ap_list* is a pointer to a list of records that consist of the fields *Ap* and *Next* (pointer to next record). *Ap_list* is used for each entry in the shortest-path table to record the distinct *Ap*'s of the paths in the entry. *Ap_counter* is a pointer to a list of records that consist of the fields *Ap, Counter* and *Next. Ap_counter* is used for each equivalence class to count the number of entries in which at least one of the paths' $A(p)$ is equal to Ap.

Algorithm: construction of the one-to-one mapping path table;

- input :

  f_1, f_2: the generators of a two-dimensional linear congruential graph G of simple form.

  $(2^i, s_2)$: the size of the graph G.

- output : the one-to-one mapping path table for G.

```
begin
  construct the shortest_path table for (0, 0);
  for x0 := 0 to 15 do
    for y := 0 to s2 - 1 do         {* for each equivalence class   *}
    begin
      Ap_counter := nil;
      for j := 0 to power(2, i-4) - 1 do
      begin                         {* for each vertex in the class *}
        x := x0 + 16 * j;
        Ap_list := nil;
        current_p := shortest_path_table[x,y].pathpt;
        while current_p <> nil do   {* for each path in the entry   *}
        begin
          Ap := get_Ap(current_p^.path);
                                    {* record the distinct Ap       *}
          if Ap is not in the Ap_list then
             append a record (Ap) to Ap_list;
        end; {of while current_p}

        current_Ap := Ap_list;      {* for each record in Ap_list   *}
        while current_Ap <> nil do
        begin
          if current_Ap^.Ap is not in Ap_counter then
             append a record (current_Ap^.Ap, 1) to Ap_counter;
                                    {* set counter to be 1          *}
          else
             increase the counter of the corresponding record by 1;
        end; {of while current_Ap}
      end; {of for j}

      Ap_best := 0;                 {* find the Ap with max. counter }
      max_ct := 0;                  {  and assign to Ap_best        *}
      current_Ap_ct := Ap_counter;
      while current_Ap_ct <> nil do
        if current_Ap_ct^.Counter > max_ct then
           Ap_best := current_Ap_ct^.Ap;

      min_len := 0;
      found := true;
      for j := 0 to power(2, i-4) -1 do
                                    {* for each vertex in the class *}
      begin
        x := x0 + 16 * j;          {* remove paths whose Ap<>Ap_best*}
        for each path p in shortest_path_table[x,y]
           if get_Ap(p) <> Ap_best then
              remove the path from the entry;

       if shortest_path_table[x,y].pathpt = nil then
        begin                       {* if all paths are removed     *}
          found := false;

          if (min_len = 0) or
             (min_len > shortest_path_table[x,y].len) then
               min_len := shortest_path_table[x,y].len;
                                    {* record the minimum length of  }
        end; {of if}                {  any path that is removed     *}
      end; {of for j}
                                    {* in the class, there are some  }
      if not found then             {  vertices whose paths are all  }
                                    {  removed                      *}
         search_new_paths(x0, y, Ap_best, min_len)
  end; {of for x0, y}
end.
```

Let P be the set of the shortest paths (of length l) from the vertex $(0,0)$ to a vertex (x,y). If every $A(p)$, $p \in P$, is not equal to *Ap_best*, we have to search a new path p' (of length greater than l) such that $A(p')$ is equal to *Ap_best*.

One of the ways is to search paths from (x,y) along its four edges and to check the original shortest-path table. If we cannot find any path whose $A(p)$ is equal to

Ap_best, the searching procedure proceeds along its four neighbors' outgoing edges. If no path is found, similarly, the searching procedure proceeds recursively until at least one path is found

This method has the following disadvantages. First, the searching procedure has to be executed once for each vertex that requires a new path. In addition, the new path founded in the way may not be as shorter as possible because the longer paths may appear earlier than the shorter ones in the searching procedure.

We propose another method that can prevent from the above disadvantages. Let (d_x, d_y) be a vertex of G, $0 \leq d_x \leq 15$, $S = \{(x, y) \mid x \bmod 16 = d_x, y = d_y\}$ be the equivalence class that contains (d_x, d_y), and $S_e = \{(x, y) \mid (x, y) \in S$, and all paths in the entry corresponding to (x, y) of the shortest-path table for $(0, 0)$ of G are removed $\}$. If S_e is not empty, for each vertex $(x, y) \in S_e$, we have to find the shortest new path p whose $A(p)$ is equal to *Ap_best*. The length of the new path is greater than *min_len*. *Ap_best* and *min_len* are defined as those in the above algorithm.

Let n_1, n_2, n_3, n_4 be the numbers of generators f_1, f_2, f_1^{-1}, f_2^{-1} in p, respectively. The following steps are taken to find the new paths for all vertices in S_e. We start the procedure with searching new paths of length $len = min\_len + 1$.

step 1: Find the possible combinations of (n_1, n_2, n_3, n_4) such that $n_1+n_2+n_3+n_4 = len$, and $5^{n_1} \cdot 9^{n_2} \cdot \bar{a}_{11}^{n_3} \cdot \bar{c}_{11}^{n_4} = Ap\_best$.

step 2: Remove impossible combinations:

Since $G'_x = G(\{f_{1_x}, f_{2_x}\}, 16)$ and $G_y = G(\{f_{1_y}, f_{2_y}\}, s_2)$ are vertex-transitive, there exist integers n_x and n_y such that for every vertex $x \in V(G'_x)$, $f_{2_x}(x) \bmod 16 = f_{1_x}^{n_x}(x)$, and for every vertex $y \in V(G_y)$, $f_{2_y}(y) \bmod s_2 = f_{1_y}^{n_y}(y)$. According to the property of edge-change, p is a path in $G(\{f_1, f_2\}, (2^r, s_2))$ from $(0, 0)$ to (x, y), only if p is a path in $G' = G(\{f_1, f_2\}, (16, s_2))$ from $(0, 0)$ to $(x \bmod 16, y)$. We can create x- and y-position tables for G' and check each combination. Let $(d_x, d_y) = (x \bmod 16, y)$. If

$$((n_1 - n_3) + (n_2 - n_4) * n_x) \bmod 16 \neq x\_position[d_x], \text{ or}$$

$$((n_1 - n_3) + (n_2 - n_4) * n_y) \bmod s_2 \neq y\_position[d_y],$$

then the combination (n_1, n_2, n_3, n_4) is not a solution.

step 3: Create possible paths for each combination (n_1, n_2, n_3, n_4), and apply these paths on the vertex $(0,0)$ of G. For each path p, if the destination of p is equal to any $(x,y) \in S_e$, then p is one of the shortest paths, that their $A(p)$'s are equal to *Ap_best*, from $(0,0)$ to (x,y) of G.

step 4: For each $(x,y) \in S_e$, if any new path is found, remove (x,y) from S_e.

step 5: If S_e is not empty, increase *len* by 1, and repeat the steps 1 to 5.

4.2.2 Construction of the Mapping Table

We now have a one-to-one mapping path table. Although the paths in the table may not be the shortest, any source vertex can check the table to find the paths to a given destination vertex.

Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 2^i) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 2^i) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k + 1$, $k \in N$. Let $G' = G(\{f_1, f_2\}, (16, s_2))$. As we mentioned, the paths from (s_x, s_y) to (d_x, d_y) of G are in exactly one of the entries corresponding to $\{(\alpha_x(d_x \bmod 16),\ \alpha_y(d_y)),\ (\alpha_x(d_x \bmod 16) + 16,\ \alpha_y(d_y)),\ \ldots,\ (\alpha_x(d_x \bmod 16) + 16 \times (2^{i-4} - 1),\ \alpha_y(d_y))\}$ where $\alpha = (\alpha_x, \alpha_y)$ is an automorphism on $V(G')$ such that $\alpha_x(s_x \bmod 16) = 0$, and $\alpha_y(s_y) = 0$. Thus we have to check at most 2^{i-4} entries to find the paths.

A faster way to find the corresponding entry is to create a mapping table. Let p be a path from $(0,0)$ to $(d_{x0}, d_{y0}) \in V(G)$, and p is also a path from $(s_x, s_y) \in V(G)$ to $(d_x, d_y) \in V(G)$. We create a mapping table, namely d_{x0}-*table*, for each source vertex except for $(0,0)$. The d_{x0}-table has 2^i rows and s_2 columns. If the content of an entry in d_xth row and d_yth column is d_{x0}, it means that the paths from (s_x, s_y) to (d_x, d_y) can be found only in the entry corresponding to $(d_{x0}, \alpha_y(d_y))$ of the one-to-one mapping path table for $(0,0)$.

The disadvantage of this way is that there are $2^i \times s_2 - 1$ tables, and each table has $2^i \times s_2$ entries. We will use the following theorems to overcome this disadvantage and create a unique size-independent mapping table.

THEOREM 4.2.2 *Let $G_y = G(\{f_{1_y}, f_{2_y}\}, s_2)$ be a linear congruential graph where $f_{1_y}(y) = y + b_2$, $f_{2_y}(y) = y + d_2$. If p is a path from the vertex 0 to a vertex $d_{y0} \in V(G_y)$, and p is also a path from a vertex $s_y \in V(G_y)$ to a vertex $d_y \in V(G_y)$, then $d_{y0} = (d_y - s_y) \bmod s_2$. (refer to Figure 4.2)*

Proof:

For any $y \in V(G_y)$, $f_{1_y}(f_{1_y}^{-1}(y)) = y$, and $f_{2_y}(f_{2_y}^{-1}(y)) = y$.
Thus $f_{1_y}^{-1}(y) = (y + (s_2 - b_2)) \bmod s_2$, and $f_{2_y}^{-1}(y) = (y + (s_2 - d_2)) \bmod s_2$.
Since p is a path from 0 to d_{y0}, $d_{y0} = 0 + R_y(p) \bmod s_2$.
Since p is a path from s_y to d_y, $d_y = (s_y + R_y(p)) \bmod s_2 = (s_y + d_{y0}) \bmod s_2$.
Therefore, $d_{y0} = (d_y - s_y) \bmod s_2$.

□

THEOREM 4.2.3 *Let $G_x = G(\{f_{1_x}, f_{2_x}\}, 2^i)$ be a linear congruential graph. Let p be a path in G_x from the vertex 0 to a vertex d_{x0}, also a path from the vertex 1 to a vertex d_{x1} and also a path from a vertex s_x to a vertex d_x. Let p' be a path in G_x from the vertex 0 to a vertex d'_{x0} and also a path from the vertex 1 to a vertex d'_{x1}, and $A(p') = A(p)$. Then $d_{x0} = (d_x - (d'_{x1} - d'_{x0}) \times s_x) \bmod 2^i$.*

Proof:

$$\begin{aligned}
d'_{x0} &= (A(p') \times 0 + R_x(p')) \bmod 2^i = R_x(p') \bmod 2^i \\
d'_{x1} &= (A(p') \times 1 + R_x(p')) \bmod 2^i = (A(p') + d'_{x0}) \bmod 2^i
\end{aligned}$$

Thus $A(p') = (d'_{x1} - d'_{x0}) \bmod 2^i$

$$\begin{aligned}
d_{x0} &= (A(p) \times 0 + R_x(p)) \bmod 2^i = R_x(p) \bmod 2^i \\
d_{x1} &= (A(p) \times 1 + R_x(p)) \bmod 2^i = (A(p) + d_{x0}) \bmod 2^i \\
d_x &= (A(p) \times s_x + R_x(p)) \bmod 2^i = (A(p) \times s_x + d_{x0}) \bmod 2^i
\end{aligned}$$

Since $A(p) = A(p') = (d'_{x1} - d'_{x0}) \bmod 2^i$,

$$d_{x0} = (d_x - A(p) \times s_x) \bmod 2^i = (d_x - (d'_{x1} - d'_{x0}) \times s_x) \bmod 2^i$$

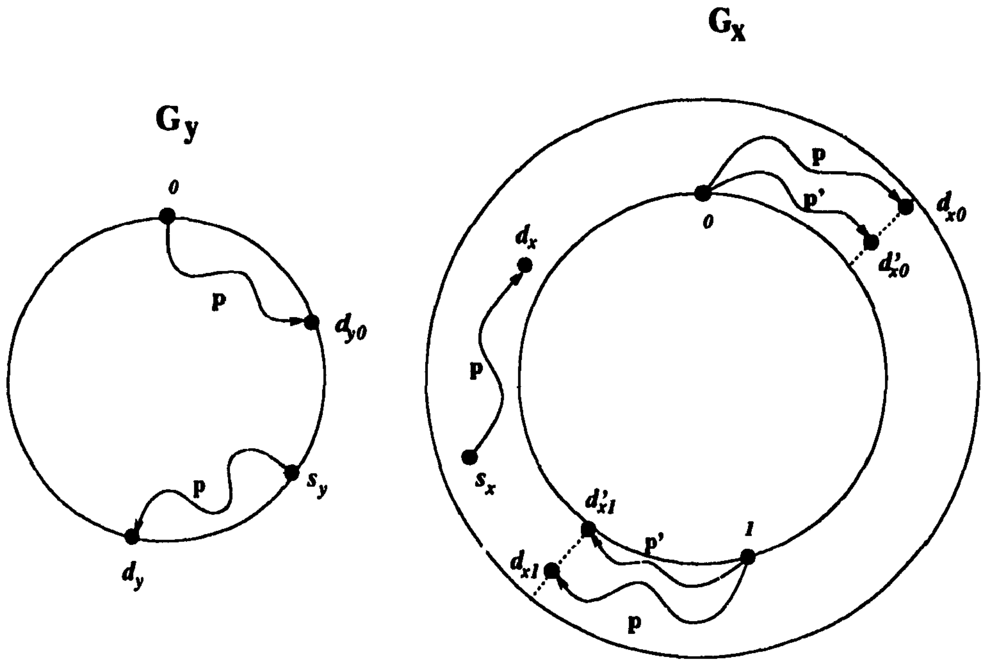


Figure 4.2: Paths with the same value of A(p)

□

Let $d'_{x1} = d_{x1}$ mod 16, d'_{x0} mod 16 = d_{x0} mod 16 (refer to Figure 4.2). According to the property of edge-change and the vertex transitivity of $G'_x = G(\{f_{1_x}, f_{2_x}\}, 16)$, since d_{x1} mod 16 = d'_{x1}, p is also a path in G'_x from the vertex 1 to the vertex d'_{x1}. We can determine d'_{x1} by checking x-position table and x-Hamiltonian table of G'_x.

$$\begin{aligned} d'_{x1} &= x\_Hamiltonian[(x\_position[1] + x\_position[d_x \bmod 16] \\ &\quad - x\_position[s_x \bmod 16]) \bmod 16] \end{aligned} \tag{4.1}$$

Thus we only need a d_{x0}-table for the source vertex (1,0) of G. This table only has 16 rows and s_2 columns. d'_{x0} can be determined by checking the entry corresponding to (d'_{x1}, d_{y0}) of the d_{x0}-table where $d_{y0} = (d_y - s_y)$ mod s_2. Then $d_{x0} = (d_x - (d'_{x1} - d'_{x0}) \times s_x)$ mod 2^i is determined. The paths from (s_x, s_y) to (d_x, d_y) are in the entry corresponding to (d_{x0}, d_{y0}) of the one-to-one mapping path table.

Table 4.3 shows the d_{x0}-table of the graph $G(\{f_1, f_2\}, (64, 9))$ where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$, and $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2,4)$. For example, to determine a path from the vertex $(s_x, s_y) = (2,3)$ to the vertex $(d_x, d_y) = (8,5)$ of G, we first find $d'_{x1} = x\_Hamiltonian[7+8-2] = x\_Hamiltonian[13] = 15$ by Equation 4.1 and $d_{y0} = (d_y - s_y) = 2$. We then check the entry corresponding to $(d'_{x1}, d_{y0}) = (15,2)$ of the d_{x0}-table to find $d'_{x0} = 22$. Finally, $d_{x0} = (d_x - (d'_{x1} - d'_{x0}) \times s_x) \bmod 2^i = (8 - (15-22) \times 2) \bmod 64 = 22$.

4.2.3 Evaluation

This global routing algorithm needs only one path table and a size-independent d_{x0}-table (of size $16 \times s_2$). However, it does not always provide the shortest paths.

We have tested some cases of size from 32×9 to 2048×9. The lengths of the paths in all tested cases exceed the diameter at most by one. In the cases of size 2048×9, the number of entries that have to find new paths is about 1000, and the length of any new path between two vertices exceeds the distance between these two vertices at most by 4 or 5.

An example in Table 4.4 illustrates their comparison.

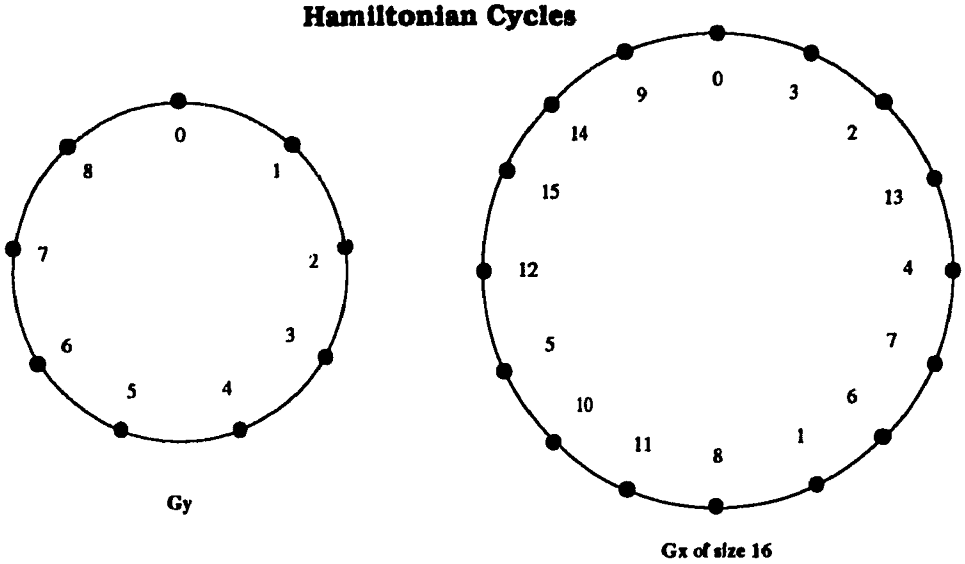
4.3 Distributed Routing

We will present two distributed routing algorithms for two-dimensional linear congruential graphs. The algorithms will use a *length table* at each vertex to decide which edge of the vertex is to be used to route messages. The length table for a vertex $\vec{u} \in V(G)$ consists of $2^i \times s_2 \times 4$ entries since there are $2^i \times s_2$ vertices, and there are 4 edges incident with $\vec{u}$. An entry corresponding to a vertex $\vec{v}$ and an edge e incident with $\vec{u}$ represents the length of the shortest path from $\vec{u}$ to $\vec{v}$ via e.

In order to route messages in a graph in the presence of faulty links, these two algorithms also use a backtracking technique similar to the algorithm in star graphs proposed by [32]. They do not require global knowledge of faults. In fact, the only knowledge required at a vertex is the state of its incident edges and the information in the length table. The routed message carries information about the path followed and the vertices that have been visited.

Both of the algorithms will route messages along one of the shortest paths if no faults are encountered. In the presence of faults, the first algorithm will always find

Hamiltonian Cycles



DxO-table

Dy0 / Dx1'	0	1	2	3	4	5	6	7	8
0	11	43	43	27	59	43	11	59	27
1	0	32	16	16	32	32	48	48	32
2	5	37	21	53	37	5	53	53	21
3	10	58	26	10	10	42	26	58	42
4	63	15	15	47	31	63	47	15	63
5	36	4	4	52	20	4	36	20	52
6	25	9	9	25	25	41	41	9	57
7	46	46	62	62	30	14	46	30	62
8	35	3	51	19	51	35	35	19	51
9	56	56	40	8	56	2	8	40	24
10	61	61	45	13	61	29	13	45	45
11	34	18	50	34	2	50	18	18	34
12	55	39	7	7	23	23	7	39	23
13	44	28	60	44	12	60	28	12	12
14	49	33	1	49	17	1	33	17	17
15	6	38	22	54	38	6	6	54	22

Let p be a path in one-to-one mapping path table. If p is a path from (1,0) to (dx1', dy0), then p is also a path from (0, 0) to (dx0', dy0).

Table 4.3: An example of the d_{x0}-table

Size of Graph	Number of entries that have to search new paths	Diameter of G	Maximum length of any new path	Max. difference in len. between any new and original paths
32 x 9	2	10	10	1
64 x 9	2	10	10	1
128 x 9	2	10	10	1
256 x9	38	10	10	3
512 x 9	107	11	12	3
1024 x 9	305	11	12	4
2048 x 9	992	12	12	4

Table 4.4: Comparison of two global routing algorithms in graphs $G(\{f_1, f_2\}, (2^i, s_2))$ of Simple Form, where

$$f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$$

$$f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2,4)$$

a path between two vertices as long as the two vertices are connected. Let $\vec{u}$ be a vertex that receives a forward message from e, one of its edges. The first algorithm allows the message to backtrack through $\vec{u}$, if all the other edges of $\vec{u}$ are either faulty or leading to a cycle.

The second algorithm may find a shorter path than the first one does under some fault conditions. However, finding of a path between two vertices is not guaranteed, even if the two vertices are connected. It allows backtracking if backtracking may lead to a shorter path.

Both algorithms will return the messages to the originating vertex if they cannot route the message. We will call the first one *conservative distributed routing algorithm*, and the second one *progressive distributed routing algorithm*. The evaluation of these two algorithms under different faulty conditions will be presented in the following subsection.

4.3.1 Construction of Length Tables

To construct a length table for a vertex $\vec{u}$ of a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$, the following data structures will be used.

For a given vertex $\vec{u} \in V(G)$, let $P = \{p \mid p$ is the shortest path from the vertex $\vec{u}$ via an edge e to a vertex $\vec{v}$, p contains no cycles, $\vec{v} \in V(G)$, and e is an edge incident with $\vec{u}\}$, $L = \{l \mid l$ is the length of p, $p \in P\}$, and *len* be the maximum of L.

A *path table* for a vertex (x_0, y_0) has s_2 rows indexed from 0 to $(s_2 - 1)$ and *len* columns. Each entry is a pointer to a list of records. A record (x, *start_edge*, *last_edge*, *next*) in a list pointed by an entry in the yth row and the lth column means that there is a path of length l from (x_0, y_0) to (x, y) in which the first edge is *start_edge*, and the last edge is *last_edge*. The path contains no cycles.

As we mentioned, the *length table* for a vertex $(x_0, y_0) \in V(G)$ has $2^i \times s_2$ rows indexed from 0 to $(2^i \times s_2 - 1)$ and 4 columns indexed from 0 to 3. Each row corresponds to a vertex of G, and each column corresponds to an edge incident with (x_0, y_0). For a vertex, we will use 0, 1, 2 and 3 to denote its incident edges corresponding to f_1, f_2, f_1^{-1} and f_2^{-1}, respectively. If the content of an entry corresponding to a vertex (x, y) and an edge e is l, then the minimum length of any path from (x_0, y_0) via e to (x, y) is l. In constructing the table, the four entries corresponding to (x_0, y_0) are initially set to be -1, and the others are set to be 0.

Algorithm: construction of a length table;

- input:

  f_1, f_2: the generators of a two-dimensional linear congruential graph G

  $(2^i, s_2)$: the size of G.

  (x_0, y_0): a vertex of G.

- output: the length_table for (x_0, y_0).

```
begin
  initialize the length_table;
  for e:=0 to 3 do          {* for each edge incident with (x0, y0) *}
  begin
      find (x, y) which is adjacent to (x0, y0) via e;
```

```
    new(new_path);
    new_path^.x := x;
    new_path^.start_edge := e;
    new_path^.last_edge := e;
    new_path^.next := nil;
    append the record (new_path) to the list pointed
     by path_table[y, 1];

    length_table[x, y, e] := 1;
end; {of for}

len := 1;
repeat
  for y:=0 to s2 - 1 do     {* for each row (y) of path_table    *}
  begin
    current := path_table[y, len];
    while current <> nil do {* for each path in the entry        *}
    begin
      for e:=0 to 3 do      {* for each edge incident with the    }
      begin                 {  destination of the path           *}
        if e is not the last edge of the path then
        begin
          find (nx, ny) which is adjacent to (current^.x, y) via e;

          if (length_table[nx, ny, current^.start_edge] =0) then
          begin             {* so far, no any other path leaving  }
                            {  from (x0, y0) along the same edge  }
                            {  has arrived (nx, ny) with a shorter}
                            {  or equal length                   *}

             new(new_path); {* extend the path                   *}
             new_path^.x := nx;
             new_path^.start_edge := current^.start_edge;
             new_path^.last_edge := e;
             append the record (new_path) to the list pointed by
               path_table[ny, len + 1];

             length_table[nx, ny, current^.start_edge] := len + 1;
          end; {of if length_table}
        end; {of if e}
      end; {of for e}
```

```
        current := current^.next;
      end; {of while current}
    end; {for y}
    len := len +1;

  until all entries in length_table are not zero;
end.
```

Similarly to the global routing algorithm described in section 4.1, the number of distinct length tables that we have to create is (2^{i-4}) for a graph of simple form, and $(2^{i-2} \times s_2)$ for a graph of complex form according to the symmetric property of the graph.

4.3.2 Conservative Algorithm

Before presenting the conservative distributed routing algorithm, we first introduce the data structure of messages routed by the algorithm.

As we mentioned, a routed message carries information about the path followed and the vertices that have been visited so far. In the description of the algorithm, we will use the following convention to denote the complete message being transmitted: $(m, d, f/w, VisitedList, Linklist)$.

m: The actual message.

d: The destination vertex.

f/w: A bit that is set for a forward transmission message and reset for a return message.

VisitedList: A list representing the vertices. Once a vertex receives a forward message, the vertex is appended to the list. Even though the message backtracks through the vertex, it will still stay in the list. Thus we can prevent from cycles in the path and ensure that any vertex will not be tried more than once.

LinkList: A list representing the edges being traversed by the message. An edge is appended to the list once the forward message traverses the edge and is deleted once the message backtracks through it.

The following is a skeleton of the algorithm in which a valid edge is defined as a non faulty edge that sends the message to a non visited vertex. Each vertex checks its length table to decide the priority of its edges once it received a message. An edge with a smaller value of the corresponding entry has a higher priority.

Skeleton of Conservative Algorithm:

1. At each vertex that receives a message $(m, d, f/w, VisitedList, LinkList)$, **if** it is the destination of the message, pick the actual message m and stop, **else if** the message is forward $(f/w = 1)$, append the vertex visited to the *VisitedList*, and append the edge just traversed to the *LinkList*, **otherwise** (it is a return message) delete the edge from the *LinkList*.

2. Select the edge that has the highest priority among its valid edges. **If** there is an edge that matches the conditions, send a message (m, d, 1, *VisitedList*, *LinkList*) forward along the edge and go to step 1, **else** (there is not any valid edge) go to the next step.

3. **If** this vertex is not the source, send a return message (m, d, 0, *VisitedList*, *LinkList*) along the last edge of the *LinkList* and go to step 1, **else** (this vertex is the source) go to the next step.

4. Halt, since there is no way to forward the message.

The algorithm allows a message to backtrack through a vertex, provided that all edges incident with the vertex are invalid. In other words, except for the edge that the forward message came from, all edges are either faulty or leading to a cycle. As we defined, a valid edge is a non faulty edge that sends the message to a non visited vertex. Thus the edge that the forward message came from is invalid, since the vertex connected by the edge is visited.

The advantage of this algorithm is that it will always find a path between two vertices as long as the two vertices are connected, since in the worst case, it executes a complete search. On the contrary, it is not flexible to route messages, even if earlier backtracking can lead to a shorter path than sending the message to the other valid edges.

For example, let $\vec{u}, \vec{v}$ be two vertices connected by an edge e_0, and let e_1, e_2, e_3 be the other edges incident with $\vec{v}$ where e_1 is faulty. Assume that the lengths of the shortest paths from $\vec{v}$ to a vertex $\vec{w}$ via e_0, e_1, e_2, e_3 are 5, 2, 8, 9, respectively. When $\vec{v}$ receives a message from $\vec{u}$, and the destination of the message is $\vec{w}$, if the conservative algorithm is applied on the network, $\vec{v}$ will choose e_2 to send the message. Thus eight message hops are required to forward the message from $\vec{v}$ to $\vec{w}$ if no more faults are encountered by the message.

Since the lengths of the shortest paths from $\vec{v}$ to $\vec{w}$ via e_0 is 5, there is an edge e' incident with $\vec{u}$ such that the length of the shortest path from $\vec{u}$ to $\vec{w}$ via e' must be 4. If $\vec{v}$ are allowed to return the message to $\vec{u}$ via e_0, and e' is valid, then the vertex $\vec{u}$ will choose e' to forward the message to $\vec{w}$. Thus only five message hops are required to send the message from $\vec{v}$ to $\vec{w}$ if no more faults are encountered by the message.

We will call this routing scheme *progressive distributed routing algorithm.* It can overcome the above disadvantage of the conservative routing algorithm. However, it may not find a path between two vertices, even if the two vertices are connected.

4.3.3 Progressive Algorithm

The formats of messages and length tables in the progressive algorithm are the same as those described in the conservative algorithm, but additional informations, *tried flags*, are required.

When a vertex receives a message, it creates four one-bit flags, namely *tried*, for the message. A flag is set if the corresponding edge has been tried, and thus the message should not be forwarded along this edge again. The flag is reset if a return message may have to backtrack through it.

Skeleton of Progressive Algorithm:

1. At each vertex that receives a message (m, d, f/w, *VisitedList*, *LinkList*), **if** it is the destination of the message, pick the actual message m and stop, **else** reset all edges "untried", and set the edge just traversed "tried".

If the message is forward ($f/w = 1$), append the vertex visited to *VisitedList*, and append the edge just traversed to *LinkList*, **otherwise** (it is a return message) delete the edge from *LinkList* and go to step 4.

2. **If** the message is forward, select the edge that has the highest priority among those edges that are not marked "tried", and **if** the edge is valid, send the message (m, d, 1, *VisitedList*, *LinkList*) forward along the edge and go to step 1, **else** set the edge "tried", reset the edge just traversed "untried" and go to the next step.

3. Select the edge that has the highest priority among those edges that are not marked "tried". The edge just traversed is one of the candidates.
   If the edge just traversed is selected, send a return message (m, d, 0, *VisitedList*, *LinkList*) along the edge and go to step 1, **else** (not the edge just traversed) **if** the edge is valid, send the message (m, d, 1, *VisitedList*, *LinkList*) forward along the edge and go to step 1, **else** (the edge is not valid) set the edge "tried" and repeat step 3 again.

4. **If** the massage is a return message ($f/w = 0$), select the edge that has the highest priority among those edges that are not marked "tried".
   If the edge selected is the last entry of *LinkList*, send a return message (m, d, 0, *VisitedList*, *LinkList*) along the edge, **else** (not backtracking) **if** the edge is valid, send the message (m, d, 1, *VisitedList*, *LinkList*) forward along the edge and go to step 1, **else** (the edge is not valid) set the edge "tried" and repeat step 4 again.

The progressive distributed routing algorithm allows a message to backtrack through a vertex, even when some edges incident with the vertex are still valid. If no faults are encountered, the algorithm routes messages along one of the shortest paths between two vertices. If one of the edges incident with a vertex $\vec{v}$ in the shortest path is faulty, routing the message along the remaining two edges incident with $\vec{v}$ probably leads to longer paths than sending a return message to previous vertex in *LinkList* and routing the message again.

However, the vertex $\vec{v}$ through which the message backtracks cannot be visited again by the message. Under a specific faulty situation such that the message must go through $\vec{v}$ to arrive the destination, the progressive algorithm cannot find a path for the message. Another problem is that after backtracking, the message may encounter another fault, and therefore the number of message hops is larger than what expected.

According to our tests, which will be introduced in the next subsection, the number of cases where the above problems occur increases with the number of faulty edges.

4.3.4 Evaluation

To evaluate the algorithms introduced in the previous subsections, we provide an empirical evaluation for two-dimensional linear congruential graphs $G(\{f_1, f_2\}, (2^i, 9))$ of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2,4)$, and $5 \leq i \leq 9$. In these cases, a *column* x_0 is defined as a set of vertices $\{(x,y) \mid x = x_0, (x,y) \in V(G)\}$, and a *row* y_0 is defined as a set of vertices $\{(x,y) \mid y = y_0, (x,y) \in V(G)\}$, therefore $V(G)$ can be partitioned into 2^i columns or 9 rows.

Since the graph G is of simple form, $G = G_x \circ G_y$, and therefore the value of x does not depend on the value of y. If a vertex (x_1, y_1) is adjacent to a vertex (x_2, y_2), then every vertex in the column x_1 is adjacent to a vertex in the column x_2, and every vertex in the row y_1 is adjacent to a vertex in the row y_2. Thus we say that *column* x_1 *is adjacent to column* x_2, and *row* y_1 *is adjacent to row* y_2.

In the graphs G_x and G_y, any two edges incident with the same vertex are disjoint. If the graph $G = G_x \circ G_y$ has n faulty columns (or rows) for $1 \leq n \leq 4$, a vertex not in the faulty columns (or rows) has at most n faulty edges, and any two vertices in the same column (or row) have the same number of faulty edges. Similarly, if G has n faulty columns and n faulty rows for $1 \leq n \leq 2$, a vertex not in the faulty columns and faulty rows has at most $2n$ faulty edges.

According to the maximum number of faulty edges of any vertex that is not isolated, we designed the cases as follows.

1. There is no fault.

2. There is a faulty vertex. The number of cases is $2^i \times 9$.

3. There is a faulty column. The number of cases is 2^i.

4. There are two faulty columns. If these two faulty columns are arbitrarily chosen, there are $C_2^{2^i} = \frac{2^i \times (2^i - 1)}{2}$ combinations. To reduce the number of cases, we choose two columns that are adjacent to the same column. Thus the number of cases is $2^i \times C_2^4 = 2^i \times 6$.

5. There are three faulty columns. To reduce the number of cases, we choose three columns that are adjacent to the same column. Thus the number of cases is $2^i \times C_3^4 = 2^i \times 4$.

6. There are four faulty columns. Similarly, we choose four columns that are adjacent to the same column. Thus the number of cases is $2^i \times C_4^4 = 2^i$. In fact, since a column is isolated by the four faulty columns, the isolated column is also faulty.

7. There is a faulty row. The number of cases is 9.

8. There are two faulty rows. These two rows are arbitrarily chosen, therefore the number of cases is $C_2^9 = 36$.

9. There are three faulty rows. To reduce the number of cases, we choose three rows that are adjacent to the same row. Thus the number of cases is $9 \times C_3^4 = 36$.

10. There are four faulty rows. Similarly, we choose four rows that are adjacent to the same row. Thus the number of cases is $9 \times C_4^4 = 9$. In fact, since a row is isolated by the four faulty rows, the isolated row is also faulty.

11. There are a faulty column and a faulty row. The column and row are arbitrarily chosen, therefore the number of cases is $2^i \times 9$.

12. There are two faulty columns and two faulty rows. If they are arbitrarily chosen, the number of combinations will be $C_2^{2^i} \times C_2^9 = \frac{2^i \times (2^i - 1)}{2} \times \frac{9 \times 8}{2}$. Let

(x, y) be a vertex of G, and its four neighbors be (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, y_4). For each neighbor of (x, y), we choose a column or a row to which it belongs. Thus there are $C_2^4 = 6$ combinations $\{x_1, x_2, y_3, y_4\}$, $\{x_1, x_3, y_2, y_4\}$, $\{x_1, x_4, y_2, y_3\}$, $\{x_2, x_3, y_1, y_4\}$, $\{x_2, x_4, y_1, y_3\}$ and $\{x_3, x_4, y_1, y_2\}$. There are $2^i \times 9$ vertices, so the number of cases is $2^i \times 9 \times 6$.

As proved in subsection 3.3.1, the linear congruential graph $G_x = G(\{f_{1x}, f_{2x}\}, 2^i)$ is vertex transitive with respect to a partition P where P is as described in subsection 3.3.1, and $G_y = G(\{f_{1y}, f_{2y}\}, 9)$ is vertex transitive. For each test case, we have to test 2^{i-4} source vertices. For each source vertex, we have to test $2^i \times 9 - 1$ destination vertices.

As the result of test case 1, given a source vertex and a destination vertex, the path decided by the progressive algorithm is always the same as the path decided by the conservative algorithm when there is no fault. The path is a shortest path between these two vertices.

The other results are shown in Tables 4.5 and 4.6, in which the *percentage of faults* is defined as the number of faulty vertices divided by the size of the graph. We observe that the progressive algorithm usually routes messages with less message hops than the conservative algorithm does when percentage of faults is lower. However, even though two vertices are connected, the number of messages that cannot be routed by the progressive algorithm increases with the percentage of faults. On the contrary, the conservative algorithm always find a path between two vertices as long as these two vertices are connected.

4.4 Finding Disjoint Paths

In this section, we will investigate the connectivity of the two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ as described in section 3.3. Since the degrees of G is 4, the connectivity of G is at most 4. We will thus propose an algorithm of finding disjoint paths for graphs of connectivity 4.

Algorithm A: Progressive distributed routing algorithm

Algorithm B: Conservative distributed routing algorithm

Size: 32 x 9

% Faulty cases	1 vertex	1 column	2 columns	3 columns	4 columns	1 row	2 rows	3 rows	4 rows	1 column & 1 row	2 columns & 2 rows
Faults (Isolated)	0.35	3.13	6.25	9.38	15.63	11.11	22.22	33.33	55.56	13.88	27.43
Both cannot forward	0.35	3.13	6.25	9.38	15.63	11.11	22.22	33.33	55.56	13.88	27.43
Only B can forward	0.00	0.00	0.01	0.36	0.00	0.00	8.29	29.89	23.69	0.02	11.30
B is faster	0.00	0.01	0.76	1.44	0.76	0.00	3.37	2.59	0.65	1.04	5.18
A is faster	0.00	0.05	0.88	1.46	0.59	0.46	5.48	6.88	0.91	1.89	8.37
A = B = Optimal	98.70	88.55	78.16	69.37	65.40	60.28	31.45	14.91	13.20	52.68	23.24
A = B > Optimal	0.95	8.26	13.95	17.99	17.62	28.15	29.19	12.40	5.97	30.49	24.32

Size: 64 x 9

% Faulty cases	1 vertex	1 column	2 columns	3 columns	4 columns	1 row	2 rows	3 rows	4 rows	1 column & 1 row	2 columns & 2 rows
Faults (Isolated)	0.174	1.56	3.13	4.69	7.81	11.11	22.22	33.33	55.56	12.50	24.89
Both cannot forward	0.174	1.56	3.13	4.69	7.81	11.11	22.22	33.33	55.56	12.50	24.89
Only B can forward	0.000	0.00	0.00	0.08	0.00	0.02	8.74	30.62	23.77	0.03	11.52
B is faster	0.000	0.15	0.39	0.49	0.37	5.84	10.22	4.80	2.46	6.53	10.71
A is faster	0.002	0.02	0.30	0.45	0.11	1.08	11.35	9.44	4.63	2.03	11.57
A = B = Optimal	99.220	93.09	87.10	82.06	79.38	54.87	25.57	10.84	8.02	50.73	21.54
A = B > Optimal	0.604	5.18	9.08	12.23	12.33	27.08	21.91	10.97	5.57	28.19	19.76

Size: 128 x 9

% Faulty cases	1 vertex	1 column	2 columns	3 columns	4 columns	1 row	2 rows	3 rows	4 rows	1 column & 1 row	2 columns & 2 rows
Faults (Isolated)	0.087	0.78	1.56	2.34	3.91	11.11	22.22	33.33	55.56	11.81	23.53
Both cannot forward	0.087	0.78	1.56	2.34	3.91	11.11	22.22	33.33	55.56	11.81	23.53
Only B can forward	0.000	0.00	0.00	0.02	0.00	0.00	8.58	32.18	24.40	0.01	11.40
B is faster	0.000	0.03	0.10	0.27	0.09	9.49	19.12	8.55	3.25	9.92	17.92
A is faster	0.005	0.05	0.31	0.34	0.13	2.09	11.23	9.21	5.99	2.57	11.80
A = B = Optimal	99.549	95.98	92.49	89.51	87.70	49.19	21.01	8.54	5.26	47.11	19.04
A = B > Optimal	0.359	3.16	5.54	7.52	8.16	28.12	17.84	8.18	5.54	28.58	16.31

Table 4.5: Comparison of two distributed routing algorithms

Algorithm A: Progressive distributed routing algorithm

Algorithm B: Conservative distributed routing algorithm

Size: 256 x 9

% \ Faulty cases	1 vertex	1 column	2 columns	3 columns	4 columns	1 row	2 rows	3 rows	4 rows	1 column & 1 row	2 columns & 2 rows
Faults (Isolated)	0.043	0 39	0 78	1.17	1.95	11.11	22 22	33 33	55 56	11.46	22.87
Both cannot forward	0 043	0.39	0.78	1.172	1.95	11.111	22 22	33 33	55.56	11.46	22 87
Only B can forward	0 000	0.00	0 00	0.005	0.00	0.005	8.89	33.64	25.85	0.01	11.62
B is faster	0.000	0 01	0 04	0.061	0.05	5.618	20 04	9 06	5.97	6 06	19 31
A is faster	0 005	0 04	0.12	0.162	0.13	1.818	12 54	9 49	6 14	2 19	12 95
A = B = Optimal	99.748	97.74	95 77	94 079	93.01	44 691	17.34	6.43	3.45	43 66	16 45
A = B > Optimal	0.204	1.81	3.28	4.522	4.86	36.757	18.97	8.05	3 04	36 62	16 80

Size: 512 x 9

% \ Faulty cases	1 vertex	1 column	2 columns	3 columns	4 columns	1 row	2 rows	3 rows	4 rows	1 column & 1 row	2 columns & 2 rows
Faults (Isolated)	0.020	0.20	0.39	0.59	0.98	11 11	22.22	33.33	55.56	11 28	22 55
Both cannot forward	0 020	0.195	0.39	0.59	0.98	11.11	22.22	33.33	55 56	11.28	22.55
Only B can forward	0.000	0 000	0.00	0.00	0.00	0 01	6.21	35.71	28 23	0.01	11.82
B is faster	0 000	0.004	0.02	0 05	0.02	6.68	14 25	10.06	6.25	6.96	20 38
A is faster	0.006	0.051	0.13	0.15	0 13	4.11	10.89	10 79	5.62	4 35	16 27
A = B = Optimal	99.860	98.744	97.66	96.73	96.13	40 65	35 51	4.83	2.34	40 12	14 02
A = B > Optimal	0 113	1 007	1.81	2.49	2.74	37.43	10.92	5.28	1.99	37 28	14 96

Table 4.6: Comparison of two distributed routing algorithms (continue)

4.4.1 Connectivity of a graph

Before investigating the connectivity of a graph $G(\{f_1, f_2\}, (2^i, s_2))$ of simple form, we first review the following theorem.

THEOREM 4.4.1 [13] **Connectivity of a Linear Congruential Graph**

Let $t \leq i$ be an integer, and $F = \{f_k \mid 1 \leq k \leq t,\ f_1$ satisfies Lemma 3.2.1, and f_k satisfies Lemma 3.2.2 if $k \geq 1\}$. Then the graph $G(F, 2^i)$ of degree $2t$ is $2t$-connected.

Let $G(\{f_1, f_2\}, (2^i, s_2))$ be a two-dimensional linear congruential graph of simple form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 2^i) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 2^i) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k + 1$, $k \in N$. Let $G_x = G(\{f_{1x}, f_{2x}\}, 2^i)$, and $G_y = G(\{f_{1y}, f_{2y}\}, s_2)$ such that $G = G_x \circ G_y$. The generators f_{1y} and f_{2y} always generate Hamiltonian cycles in G_y. The cycle structure of G is thus determined by G_x, i.e., G has the same cycle structure as G_x. Because f_{1x} and f_{2x} satisfy Lemma 3.2.1 and 3.2.2, respectively, according to the above theorem, the connectivity of G_x is 4. Then the connectivity of G is also 4.

For a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ of complex form where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, b_2)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ c_{21} & 1 \end{pmatrix} + (d_1, d_2)$, $\gcd(b_1, 2^i) = 1$, $d_1 = 2\beta$, $\gcd(\beta, 2^i) = 1$, $\gcd(b_2, s_2) = 1$, $\gcd(d_2, s_2) = 1$, and $s_2 = 4k + 1$, $k \in N$, its cycle structure may be more complex than that of simple form, but the following theorem provides a sufficient condition such that a two-dimensional linear congruential graph has connectivity $2t$.

THEOREM 4.4.2 [1] **Connectivity of a Two-Dimensional Linear Congruential Graph**

Let $G(F, (2^i, s_2))$ be a two-dimensional linear congruential graph where $F = \{f_j \mid 1 \leq j \leq t,\ t \in N\}$, and $\vec{u}$ be any vertex of G. If the set of generators has following properties:

1. *f_1 generates a Hamiltonian cycle.*

2. *$f_j(\vec{u}) \bmod (2^i, s_2) \neq f_j^{-1}(\vec{u}) \bmod (2^i, s_2)$.*

3. The generators can be arranged in a way that

- *f_{j+1} partitions every cycle generated by f_j into two disjoint sets,*
- *any two consecutive vertices $\vec{u}$ and $f_j(\vec{u})$ on the cycle generated by f_j are on the different cycles generated by f_{j+1}, and*
- *$f_j(f_j(\vec{u}))$ and $\vec{u}$ are on the same cycle generated by f_{j+1}.*

then the graph $G(F,(2^i,s_2))$ of even degree has connectivity $2t$.

Therefore, if we properly choose the generators f_1 and f_2 according to Theorem 4.4.2, we will have a graph of connectivity 4.

4.4.2 Algorithm of Finding Disjoint Paths

As proved in [13] and [1], for a two-dimensional linear congruential graph $G(\{f_1,f_2\},(2^i,s_2))$ of simple form or graph of complex form that satisfies Theorem 4.4.2, we can find four disjoint paths for each pair of vertices $\vec{v}_1$, $\vec{v}_2 \in V(G)$.

Let C_1, C_2 be two cycles generated by f_2. If both $\vec{v}_1$ and $\vec{v}_2$ are on the same cycle C_1, then there is a set of disjoint paths:

path 1: $\vec{v}_1$, $f_1(\vec{v}_1)$, part of C_2 to $f_1(\vec{v}_2)$ not containing $f_1^{-1}(\vec{v}_2)$, $\vec{v}_2$;

path 2: $\vec{v}_1$, $f_1^{-1}(\vec{v}_1)$, part of C_2 to $f_1^{-1}(\vec{v}_2)$, $\vec{v}_2$;

path 3,4: the two parts of C_1.

If $\vec{v}_1 \in C_1$ and $\vec{v}_2 \in C_2$, then there is a set of disjoint paths:

path 1: $\vec{v}_1$, $f_1(\vec{v}_1)$, part of C_2 to $\vec{v}_2$ not containing $f_1^{-1}(\vec{v}_1)$;

path 2: $\vec{v}_1$, $f_1^{-1}(\vec{v}_1)$, part of C_2 to $\vec{v}_2$ not containing $f_1(\vec{v}_1)$;

path 3: $\vec{v}_1$, part of C_1 to $f_1(\vec{v}_2)$ not containing $f_1^{-1}(\vec{v}_2)$, $\vec{v}_2$;

path 4: $\vec{v}_1$, part of C_1 to $f_1^{-1}(\vec{v}_2)$, $\vec{v}_2$.

In this way, the disjoint paths are easily decided. However, they obviously may be much longer than the shortest path, and one path may be much longer than some other paths. Since there is not only one set of disjoint paths, there exist two strategies to choose a set of disjoint paths.

1. We keep at least one of the shortest paths, even if the remaining disjoint paths are much longer than the chosen one.

2. We may not keep a shortest path, but each of them is not longer than an upper bound, and therefore the difference of lengths between any two disjoint paths is restricted.

We prefer the second strategy since it can find an upper bound for each path in the set of disjoint paths, and we conjecture that it will be faster than the first strategy. Thus we propose an algorithm for the second strategy.

In our algorithm, there is a recursive procedure *Extensive_Search.* It is called for four given pairs of vertices if a set of disjoint paths cannot be found by merely checking their shortest paths. We will see it in detail later.

The algorithm will find a set of disjoint paths between two vertices $\vec{u}$ and $\vec{v}$ in which each path is not longer than ($min\_len + 4 + 2 \times max\_level$) where min_len is the distance between $\vec{u}$ and $\vec{v}$, and max_level is the maximum level of recursive calls for *Extensive_Search.*

When more than one set of disjoint paths are found, the shortest one among them is chosen. Let $P = \{p_1, p_2, p_3, p_4\}$ and $P' = \{p'_1, p'_2, p'_3, p'_4\}$ be two set of disjoint paths between two vertices. The lengths of paths in P and P' are sorted to be (l_1, l_2, l_3, l_4) and (l'_1, l'_2, l'_3, l'_4) in the ascendant order. Whether or not *set P is shorter than set P'* is defined as follows.

Starting from $j = 1$, if $l_j < l'_j$, then P is shorter than P', else if $l_j > l'_j$, then P' is shorter than P, otherwise j is increased by 1, and the next pair of (l_j, l'_j) is compared until $j > 4$. For example, if $(l_1, l_2, l_3, l_4) = (3, 4, 7, 8)$ and $(l'_1, l'_2, l'_3, l'_4) = (3, 5, 6, 7)$, then P is shorter than P' since $4 < 5$.

Our algorithm can be described in pseudo-code as follows. (refer to Figure 4.3).

Algorithm: Finding Disjoint Paths

- input :

  f_1, f_2: the generators of a two-dimensional linear congruential graph G of connectivity 4.

  $(2^i, s_2)$: the size of G.

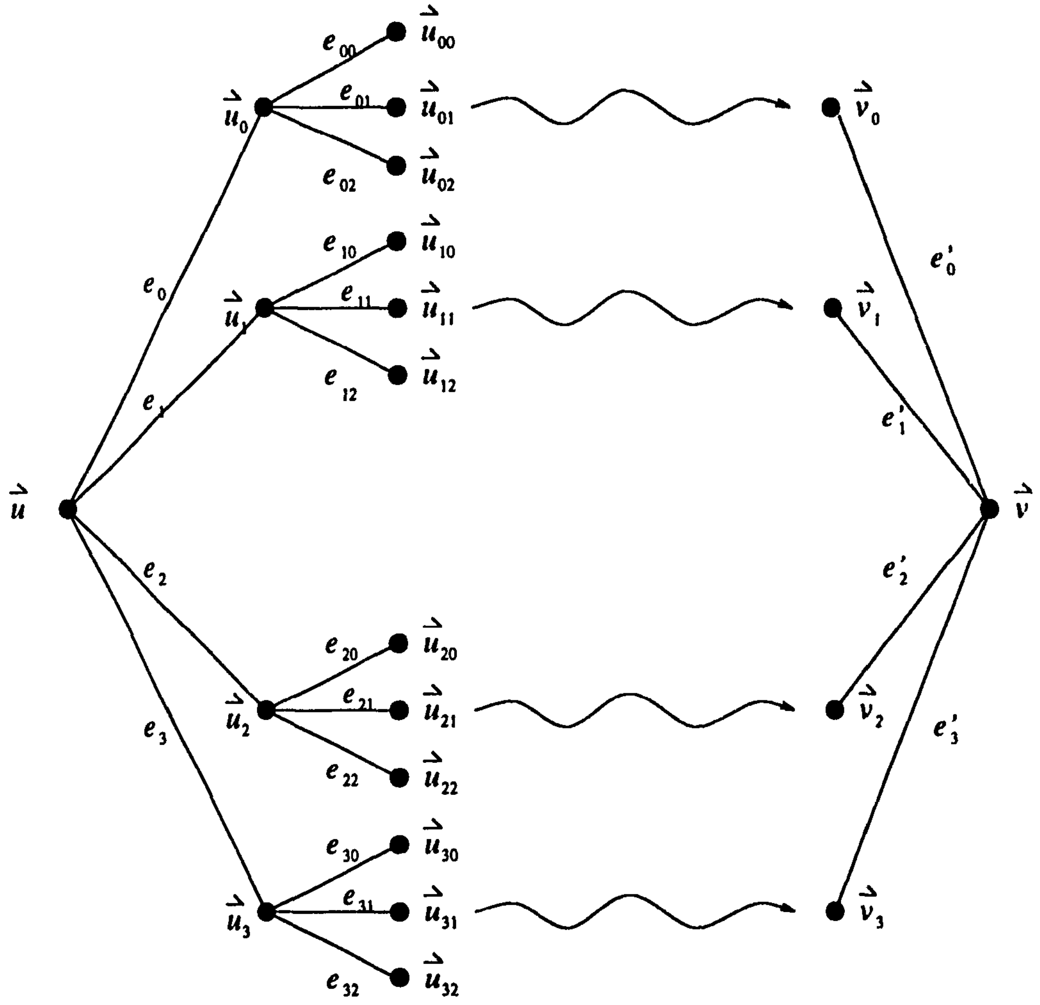


Figure 4.3: Disjoint paths

$\vec{u}, \vec{v}$: two vertices of G.

- output : the shortest set $P = \{p_1, p_2, p_3, p_4\}$ of disjoint paths between $\vec{u}$ and $\vec{v}$ where $length(p_j) \leq (min\_len + 4 + 2 \times max\_level)$, $0 \leq j \leq 3$.

```
begin
    create the shortest_path tables;
    find the set $S = \{\{(\vec{u}_0, \vec{v}_0), (\vec{u}_1, \vec{v}_1), (\vec{u}_2, \vec{v}_2), (\vec{u}_3, \vec{v}_3)\} \mid$
            $\vec{u}_0, \vec{u}_1, \vec{u}_2, \vec{u}_3 \in \{f_1(\vec{u}), f_2(\vec{u}), f_1^{-1}(\vec{u}), f_2^{-1}(\vec{u})\}$,
            $\vec{v}_0, \vec{v}_1, \vec{v}_2, \vec{v}_3 \in \{f_1(\vec{v}), f_2(\vec{v}), f_1^{-1}(\vec{v}), f_2^{-1}(\vec{v})\}$,
            $\vec{u}_0 \neq \vec{u}_1 \neq \vec{u}_2 \neq \vec{u}_3$, and $\vec{v}_0 \neq \vec{v}_1 \neq \vec{v}_2 \neq \vec{v}_3$. };
                                        {* $max(|S|) = 24$ *}
    $found$ := false;
    $max\_level$ := 0;
    while not $found$ do
    begin
        for each element $S_n \in S$
        begin
            call $find\_disjoint\_paths(S_n, max\_level, found', p'_0, p'_1, p'_2, p'_3)$;
                                        {* For $0 \leq j \leq 3$, $p'_j$ is a path from $\vec{u}_j$ to $\vec{v}_j$.
                                        If a set of disjoint paths $\{p'_0, p'_1, p'_2, p'_3\}$ are
                                        found, then $found'$ is assigned to be true,
                                        else false. * }
            if $found'$ then
            begin
                for $j$ := 1 to 3 do
                    $p_j := e_j \cdot p'_j \cdot e'_j$;
                                        {* We use "$\cdot$" to denote concatenation.
                                        $e_j$ is the edge incident with $\vec{u}$ and $\vec{u}_j$. $e'_j$ is
                                        the edge incident with $\vec{v}$ and $\vec{v}_j$.
                                        $\{p_0, p_1, p_2, p_3\}$ is a set of disjoint paths between $\vec{u}$ and $\vec{v}$. * }
                $found$ := true;
            end; { of if }
        end; { of for }
        if $found$ then
```

```
            choose the shortest set $\{p_0, p_1, p_2, p_3\}$ from all $S_n$'s;
        else
            $max\_level := max\_level + 1$;
    end; { of while }
end.
```

Procedure: Find_Disjoint_Paths$(S_n, max\_level, found', p'_0, p'_1, p'_2, p'_3)$;

- input :

  S_n: an element of S.

  max_level: the maximum level that the recursive procedure *Extensive_search* is allowed to be called.

- output :

  p'_1, p'_2, p'_3, p'_4: a set of disjoint paths where p'_j is a path from $\vec{u}_j$ to $\vec{v}_j$, $0 \leq j \leq 3$.

  $found'$: whether the set $\{p'_1, p'_2, p'_3, p'_4\}$ is found or not.

```
begin
    find the set $P' = \{\{p'_0, p'_1, p'_2, p'_3\} \mid p'_j$ is a shortest path from $\vec{u}_j$ to $\vec{v}_j$, $0 \leq j \leq 3$,
                and $p'_0, p'_1, p'_2, p'_3$ are disjoint. };
    if $P' \neq \phi$ then
    begin
        $found'$ := true;
        choose any element $\{p'_0, p'_1, p'_2, p'_3\} \in P'$;
                                                {* Each element has the same length. * }
    end { of if }
    else if $max\_level = 0$ then
            $found'$ := false
        else
        begin
            $visited\_list := \phi$;
```

```
                                        {* $visited\_list$ will be used to record the vertices that have been visited by any one of the $leading\ path$. Now, $e_j$ is a leading path where $0 \le j \le 3$. *}
            $cur\_level := 1$;
            call $extensive\_search$ ($S_n$, $max\_level$, $cur\_level$, $visited\_list$, $found''$,
                                 $p'_0, p'_1, p'_2, p'_3$);
            $found' := found''$;
        end; { of if $max\_level \neq 0$ }
end.
```

Procedure: Extensive_Search(S_n, *max_level*, *cur_level*, *visited_list*, $found''$, p'_0, p'_1, p'_2, p'_3);

input : S_n, *max_level*, *cur_level*, *visited_list*.

output : $found''$, p'_0, p'_1, p'_2, p'_3.

```
begin
    $visited\_list := visited\_list \cup \{\vec{u}_0, \vec{u}_1, \vec{u}_2, \vec{u}_3\}$;
                                        {* Note: $S_n = \{(\vec{u}_0, \vec{v}_0), (\vec{u}_1, \vec{v}_1), (\vec{u}_2, \vec{v}_2), (\vec{u}_3, \vec{v}_3)\}$ * }
    find the set $P'' = \{\{p''_0, p''_1, p''_2, p''_3\} \mid p''_j$ is a shortest path from $\vec{u}_{jk}$ to $\vec{v}_j$,
            where $\vec{u}_{jk}$ is a vertex adjacent to $\vec{u}_j$, but $\vec{u}_{jk} \neq \vec{u}$,
            $0 \le j \le 3$, $0 \le k \le 2$,
            $p''_j$ does not visit any vertex in $visited\_list$, and
            $p''_0, p''_1, p''_2, p''_3$ are disjoint. };
    if $P'' \neq \phi$ then
    begin
        $found''$ := true;
        choose the shortest set $\{p''_0, p''_1, p''_2, p''_3\}$ from $P''$;
        for $j$ := 1 to 3 do
            $p'_j := e_{jk_j} \cdot p''_j$;
                                        {* $p''_j$ originates from $\vec{u}_{jk_j}$ where $0 \le j \le 3$, $0 \le k_j \le 2$.
                                        $e_{jk}$ is the edge from $\vec{u}_j$ to $\vec{u}_{jk}$ where $0 \le k \le 2$. * }
    end { of if $P''$ }
```

```
  else if $max\_level > cur\_level$ then
    begin
      find the set $S' = \{\{(\vec{u}_{0k_0}, \vec{v}_0), (\vec{u}_{1k_1}, \vec{v}_1), (\vec{u}_{2k_2}, \vec{v}_2), (\vec{u}_{3k_3}, \vec{v}_3)\} \mid$
      $0 \leq k_0, k_1, k_2, k_3, j, l \leq 3$, $\vec{u}_{jk_j}$ does not belong to $visited\_list$,
      and $\vec{u}_{jk_j} \neq \vec{v}_l$ if $l \neq k$. };
                                        {* $max(|S'|) = 3^4$ *}
      for each element $S'_n \in S'$
      begin
        $cur\_level := cur\_level + 1$;
        call $extensive\_search$ ($S'_n$, $max\_level$, $cur\_level$, $visited\_list$, $found'''$,
                                $p''_0, p''_1, p''_2, p''_3$);
        if $found'''$ then
        begin
          for $j := 1$ to 3 do
            $p'_j := e_{jk_j} \cdot p''_j$;
          $found'' :=$ true;
        end; { of if $found'''$ }
      end; { of for }
      if $found''$ then
        choose the shortest set $\{p'_0, p'_1, p'_2, p'_3\}$ from all $S'_n$'s;
    end; { of if $max\_level$ }
end;
```

As we mentioned, if the length of the shortest paths between $\vec{u}$ and $\vec{v}$ is min_len, and a set P of disjoint paths between them is found after max_level levels of recursive calls, then the length of each path in the set is less than or equal to $(min\_len + 4 + 2 \times max\_level)$. In the following proof, we will use $len(p)$ to denote the length of a path p. Figure 4.4 illustrates the proof.

Proof:

- Let $max\_level = 0$, i.e., the procedure *Extensive_search* is not called.

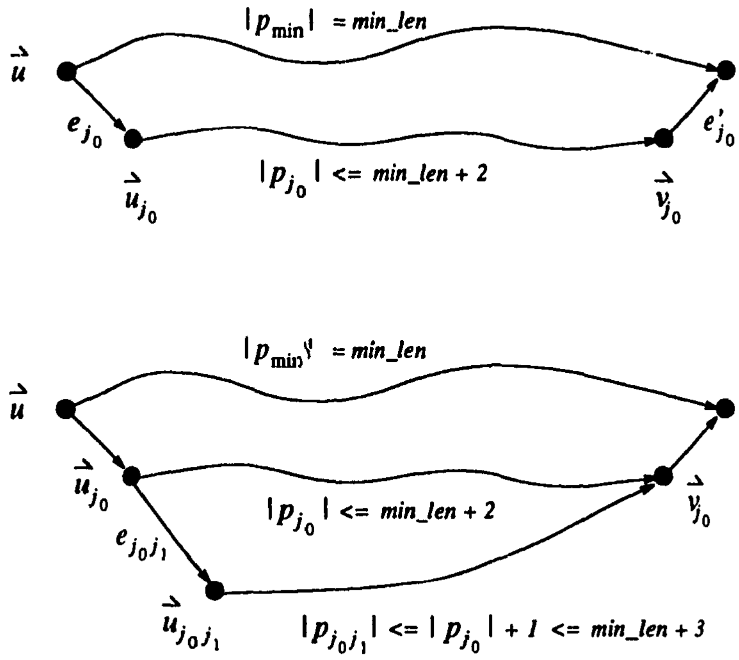


Figure 4.4: Upper bound on lengths of disjoint paths

Let p_{j_0} be a shortest path from $\vec{u}_{j_0}$ to $\vec{v}_{j_0}$ where $\vec{u}_{j_0}$ is a vertex adjacent to $\vec{u}$, $\vec{v}_{j_0}$ is a vertex adjacent to $\vec{v}$, and $0 \le j_0 \le 3$. Then $len(p_{j_0}) \le min\_len + 2$.

If $p = e_{j_0} \cdot p_{j_0} \cdot e'_{j_0}$ is a path in the set P where e_{j_0} is the edge from $\vec{u}$ to $\vec{u}_{j_0}$, and e'_{j_0} is the edge from $\vec{v}_{j_0}$ to $\vec{v}$, then $len(p) \le 1 + (min\_len + 2) + 1 = min\_len + 4$.

- Let $max\_level = 1$.

  Let $P_{j_0j_1}$ be a shortest path from $\vec{u}_{j_0j_1}$ to $\vec{v}_{j_0}$ where $\vec{u}_{j_0j_1}$ is a vertex adjacent to $\vec{u}_{j_0}$, $\vec{u}_{j_0j_1} \neq \vec{u}$, and $0 \le j_1 \le 2$. Then $len(p_{j_0j_1}) \le len(p_{j_0}) + 1 \le (min\_len + 2) + 1$.

  If $p = e_{j_0} \cdot e_{j_0j_1} \cdot p_{j_0j1} \cdot e'_j$ is a path in the set P where $e_{j_0j_1}$ is the edge from $\vec{u}_{j_0}$ to $\vec{u}_{j_0j_1}$, then $len(p) \le 1 + 1 + len(p_{j_0j_1}) + 1 \le (min\_len + 3) + 3 = min\_len + 4 + 2 \times max\_level$.

Induction Step:

When $max\_level = n$, assume $p_{j_0j_1 \dots j_n}$ is a shortest path from $\vec{u}_{j_0j_1 \dots j_n}$ to $\vec{v}_{j_0}$ where $0 \le j_1, \dots, j_n \le 2$, $\vec{u}_{j_0j_1 \dots j_k}$ is a vertex adjacent to $\vec{u}_{j_0j_1 \dots j_{k-1}}$, $\vec{u}_{j_0j_1 \dots j_k} \neq \vec{u}_{j_0j_1 \dots j_{k-2}}$, $0 \le k \le n$, and assume $len(p_{j_0j_1 \dots j_n}) \le (min\_len + 2) + n$.

Max. Level of Recursive Calls Size	0	1	2
32x 9	X	OK	
64x 9	X	OK	
128 x 9	X	X	OK
256 x 9	X	X	OK
512 x 9	X	X	OK

X : It cannot find a set of four disjoint paths for some pairs of vertices.

OK : It finds a set of four disjoint paths for any pair of vertices.

Table 4.7: Test results of finding disjoint paths

When $max\_level = n + 1$, $len(p_{j_0j_1\ldots j_nj_{n+1}}) \leq len(p_{j_0j_1\ldots j_n}) + 1 \leq (min\_len + 2 + n) + 1$. If $p = e_{j_0} \cdot e_{j_0j_1} \cdots e_{j_0j_1\ldots j_{n+1}} \cdot p_{j_0j_1\ldots j_nj_{n+1}} \cdot e'_{j_0}$ is a path in the set P, then $len(p) \leq (n + 3) + (min\_len + n + 3) = (min\_len + 4) + 2 \times (n + 1)$.

□

4.4.3 Evaluation

Table 4.7 shows the maximum level of recursive calls required for any two vertices of $G(\{f_1, f_2\}, (2^i, s_2))$ to find a set of four disjoint paths between them where $f_1(x, y) = (x, y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3, 1)$, $f_2(x, y) = (x, y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2, 4)$, $5 \leq i \leq 9$. For example, when G is of size 512×9, the maximum level of recursive calls is 2. Let min_len be the distance between two vertices $\vec{u}$ and $\vec{v}$ of G. Then the length of each path in the set of disjoint paths between $\vec{u}$ and $\vec{v}$ does not exceed $(min\_len + 4 + 2 \times 2) = (min\_len + 8)$. In other words, the difference of lengths between any two disjoint paths is at most 8.

4.5 Broadcasting

4.5.1 Introduction and Definition

Broadcasting in a network is an information dissemination process in which a single node, an originator, sends a given message to all other nodes of the network by placing calls over the communication links. The process is expected to be completed as quickly as possible under the following constraints:

- each call involves only two nodes,
- each call requires one unit of time,
- in one unit of time, a node can place a call along one of its links and, in the meantime, accept calls coming along the other links (this constraint is used to simulate a full duplex network system),
- a node can only call the nodes that are directly connected to it by a communication link.

We will study the broadcasting in networks based on two-dimensional linear congruential graphs. The nodes of the network will correspond to the vertices, and the communication links correspond to the edges of the graph.

DEFINITION 4.5.1 [28] Broadcast Time of a Vertex and Broadcast Time of a Graph

Given a connected graph G and a message originator, vertex u, the broadcast time $b(u)$ of the vertex u *is defined to be the minimum number of time units required to complete broadcasting from vertex u, and the* broadcast time $b(G)$ of the graph G *is defined to be the maximum broadcast time of any vertex u of G.*

Let a vertex u be an originator in a graph G of size n. Since during each time unit the number of vertices that received the message can at most double, $b(u) \geq \lceil \log_2 n \rceil$, and thus $b(G) \geq \lceil \log_2 n \rceil$.

A complete graph K_n has $b(K_n) = \lceil \log_2 n \rceil$, but its degree is $n - 1$. Actually, after removing some edges of K_n, we can still have a graph with n vertices and $b(G)$ equal to $\lceil \log_2 n \rceil$ [28]. For example, if $n = 2^r$, and $r \in N$, we can remove edges from K_n to have the resulting subgraph G of K_n isomorphic to an r-dimensional hypercube. Therefore, $b(G) = r = \log_2 n$. An r-dimensional hypercube is of size $n = 2^r$ and degree r, and its broadcast time is $r = \log_2 n$ [28]. A graph with 2^r vertices and broadcast time r must be of degree at least r since the originator must place r calls. Thus an r-dimensional hypercube has minimum number of edges for broadcasting.

DEFINITION 4.5.2 [28] a Minimum Broadcast Graph

A graph G of size n having the minimum number of edges and $b(G) = \lceil \log_2 n \rceil$ is called a minimum broadcast graph.

An r-dimensional hypercube is a minimum broadcast graph. However, its degree r depends on its size 2^r. Since we focus our research on the g·aphs of degree 4, we will review the upper bound on broadcast time for some well known graphs of degree 4.

Let $b(n, \Delta)$ be the minimum of $b(G)$ over all graphs G with n vertices and maximum degree Δ. It is shown in [33] that $b(n, 4) \geq 1.1374 \log_2 n$, and if n is a power of 4, $b(n, 4) \leq 1.625 \log_2 n + 2.25$.

It is proved in [28] that an undirected de Bruijn graph $UB(d, D)$ of degree $2d$, diameter D (size $n = d^D$) has the broadcast time $b(UB(d, D)) = \frac{d+1}{2} D + \frac{d}{2}$, and an undirected Kautz graph $UK(d, D)$ of degree $2d$, diameter D (size $n = (d+1)(d^{D-1})$) has the broadcast time $b(UK(d, D)) = \frac{d+1}{2} D + \frac{2d-1}{2}$. If $d = 2$ (i.e., degree 4), then $b(UB(d, D)) = 1.5D + 1 = 1.5 \log_2 n + 1$, and $b(UK(d, D)) = 1.5D + 1.5 < 1.5 \log_2 n + 1$.

A k-star graph has $k!$ vertices and degree $k - 1$. It is shown in [30] that the broadcast time $b(k\text{-star}) = \sum_{i=2}^{k}(\lceil \log_2 i \rceil + 1)$. A 5-star graph is of degree 4, and its broadcast time $b(5\text{-star}) = 12$.

We will now describe a broadcasting algorithm for two-dimensional linear congruential graphs of degree 4. The broadcast times of graphs with typical generators of simple or complex form will be investigated, and they will be evaluated based on the symmetric properties of graphs. Finally, they will be compared with those of de Bruijn graphs, Kautz graphs and the 5-star graph.

4.5.2 Broadcasting Algorithm

Algorithm

Since the generator f_1 of a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, s_2))$ as described in Section 3.3 generates a Hamiltonian cycle, we may consider the cycle to be a union of *sections* where each section consists of $2r + 1$ contiguous vertices on the Hamiltonian cycle for some integer $r \geq 1$, and any two sections have at most an end point in common. The vertices of a section are connected by the edges generated by f_1. We will say that r is the *radius* of the section. There are $\lceil \frac{2^i \times s_2}{2r} \rceil$ sections.

In the algorithm, when an originator $\vec{u} \in V(G)$ broadcasts a message, the vertex $\vec{u}$ is considered to be at the center of a corresponding section, i.e., the section consists of vertices $\vec{u}$, $f_1^j(\vec{u})$ and $f_1^{-j}(\vec{u})$ for $1 \leq j \leq r$. We will call the vertices $f_1^r(\vec{u})$ and $f_1^{-r}(\vec{u})$ the *end points* of the section. The message is sent to every vertex of the section along the f_1 and f_1^{-1} edges. After a vertex of the section received the message, it also sends the message along its f_2 and f_2^{-1} edges. If a vertex for the first time receives the message, and the message is received along its f_2 or f_2^{-1} edge, then the vertex is considered to be at the center of a corresponding section, and the message is broadcasted in a similar way.

If the intersection of any two adjacent sections is their end point, then the broadcast time of the vertex $\vec{u}$ is $t_1 + t_2$ where t_1 is the number of time units required to broadcast the message from $\vec{u}$ to all centers of $\frac{n}{2r}$ sections ($n = 2^i \times s_2$), and t_2 is the number of time units required to broadcast the message from the center of a section to all vertices of the section.

A message broadcasted by the algorithm contains a *counter* which is used to distinguish whether the message is sent to the end points of a section.

The following is the skeleton of the algorithm. An illustration can be found in Figure 4.5.

1. When an originator, vertex $\vec{u}$, wants to broadcast a message, the *counter* is assigned to be r and is appended to the message. The vertex $\vec{u}$ sends the message to its adjacent vertices in the following order: $f_1(\vec{u})$, $f_1^{-1}(\vec{u})$, $f_2(\vec{u})$, $f_2^{-1}(\vec{u})$.

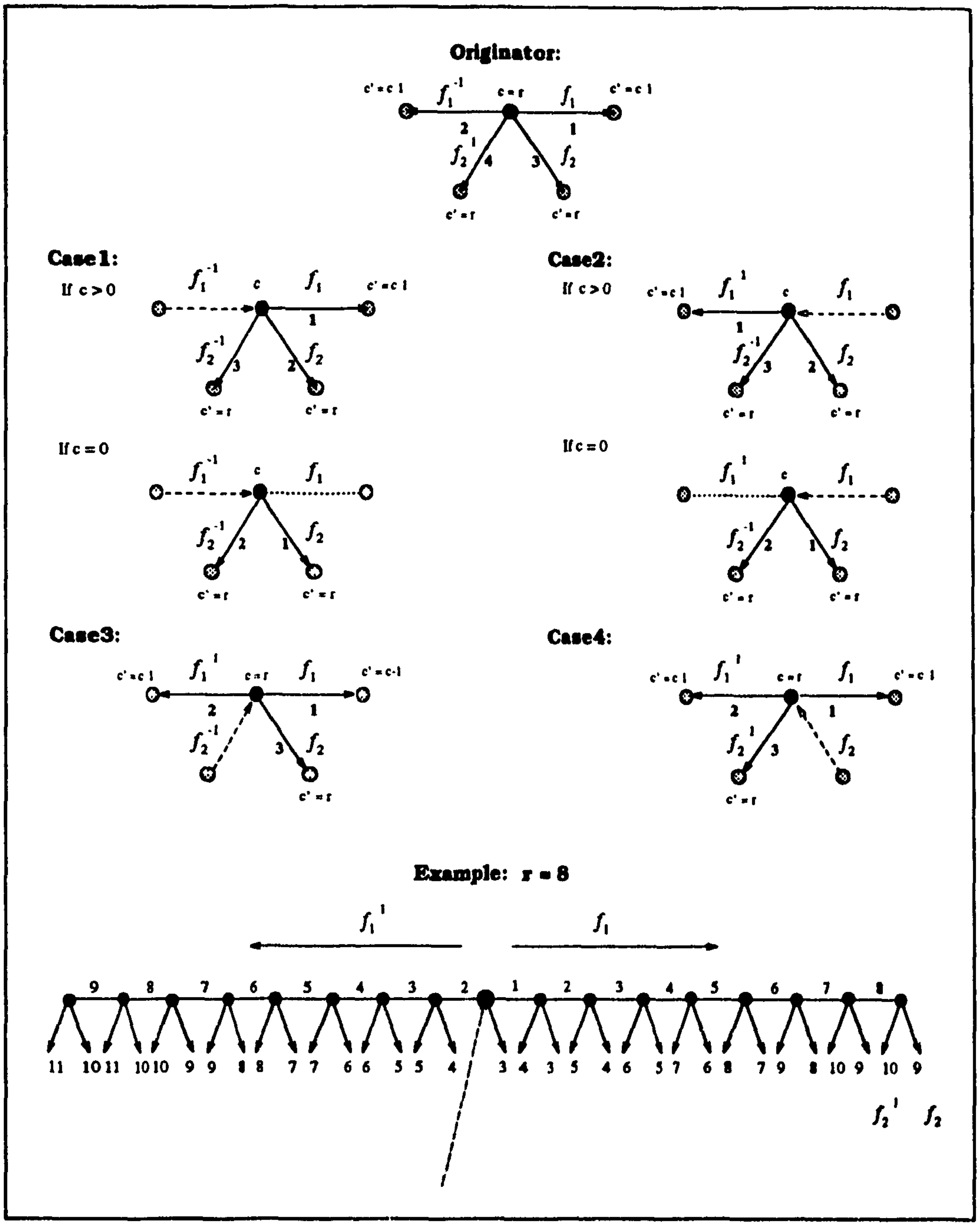


Figure 4.5: Broadcasting Algorithm

2. When a vertex $\vec{v}$ receives a message, if the message has been received before, it discards the message, otherwise it sends the message based on the adjacent vertex from which the message is received.

   (a) If the message is received from $f_1^{-1}(\vec{v})$,
   the *counter* of the message is decreased by 1, and if *counter* > 0, the vertex $\vec{v}$ sends the message to its adjacent vertices in the following order: $f_1(\vec{v})$, $f_2(\vec{v})$, $f_2^{-1}(\vec{v})$, otherwise (i.e., *counter* $= 0$) it sends the message to $f_2(\vec{v})$ and then to $f_2^{-1}(\vec{v})$.

   (b) If the message is received from $f_1(\vec{v})$,
   the *counter* of the message is decreased by 1, and if *counter* > 0, the vertex $\vec{v}$ sends the message to its adjacent vertices in the following order: $f_1^{-1}(\vec{v})$, $f_2(\vec{v})$, $f_2^{-1}(\vec{v})$, otherwise (i.e., *counter* $= 0$) it sends the message to $f_2(\vec{v})$ and then to $f_2^{-1}(\vec{v})$.

   (c) If the message is received from $f_2^{-1}(\vec{v})$,
   the *counter* of the message is assigned to be r, and the vertex $\vec{v}$ sends the message to its adjacent vertices in the following order: $f_1(\vec{v})$, $f_1^{-1}(\vec{v})$, $f_2(\vec{v})$.

   (d) If the message is received from $f_2(\vec{v})$,
   the *counter* of the message is assigned to be r, and the vertex $\vec{v}$ sends the message to its adjacent vertices in the following order: $f_1(\vec{v})$, $f_1^{-1}(\vec{v})$, $f_2^{-1}(\vec{v})$.

The reason why a vertex $\vec{v}$ always first sends the message to $f_1(\vec{v})$ or $f_1^{-1}(\vec{v})$ and then to $f_2(\vec{v})$ or $f_2^{-1}(\vec{v})$ is because all vertices of a section can complete their calls in $(r+3)$ time units after the center of the section received the message. If the reverse order is used, i.e., a vertex $\vec{v}$ always first sends the message to $f_2(\vec{v})$ or $f_2^{-1}(\vec{v})$ and then to $f_1(\vec{v})$ or $f_1^{-1}(\vec{v})$, it requires $(3r+2)$ time units for all vertices of a section to complete their calls after the center of the section received the message.

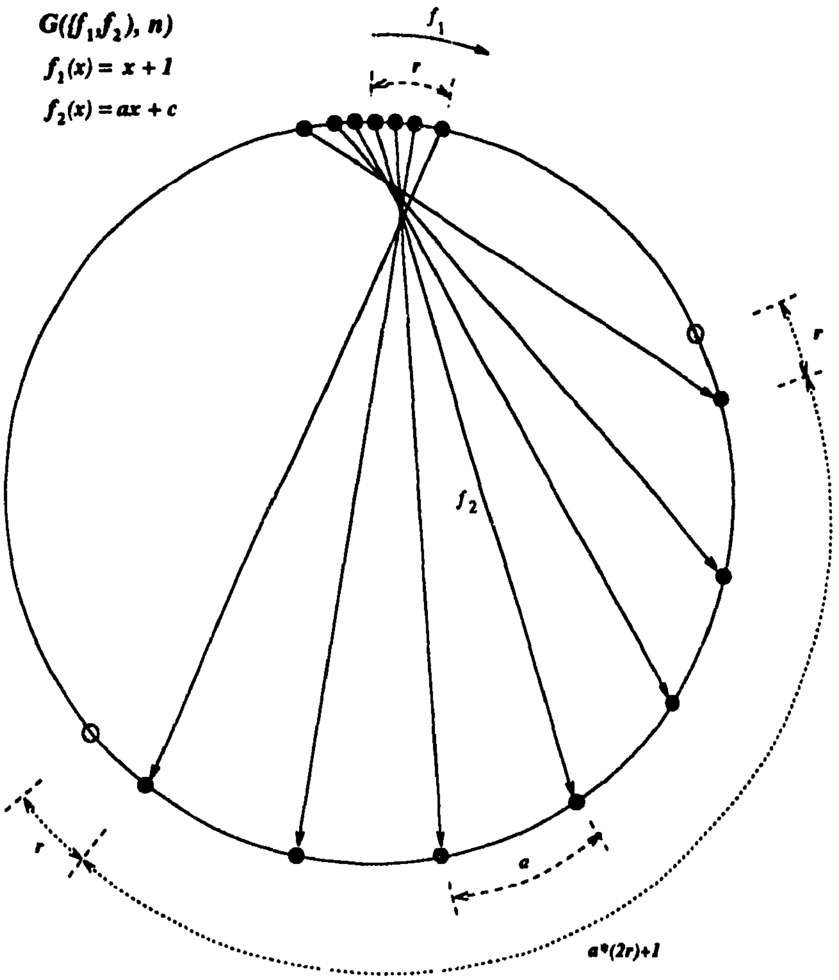


Figure 4.6: Broadcasting in a one-dimensional graph containing $f_1(x) = x + 1$

Upper bound on $b(G)$ of a one-dimensional graph G containing $f_1(x) = x+1$

Before examining the broadcast time of a two-dimensional linear congruential graph obtained by the algorithm, we first study the broadcast time of a one-dimensional linear congruential graph $G(\{f_1, f_2\}, n)$ where $f_1(x) = x + 1$, $f_2(x) = ax + b$, and $a, b \in N$. The following theorem will prove that the broadcasting in the above one-dimensional graph G can be done in time $O(\log_2 n)$ (refer to Figure 4.6).

THEOREM 4.5.1 Upper Bound on Broadcast Time of a One-dimensional Graph Containing $f_1(x) = x + 1$

Let $G(\{f_1, f_2\}, n)$ be a one-dimensional linear congruential graph where $f_1(x) = x+1$ (generating a Hamiltonian cycle), $f_2(x) = ax+b$, and let $b(G)$ be the broadcast time of the graph G obtained by the above algorithm. Then $b(G) < \lceil\frac{\log_2 n}{\log_2 a}\rceil(\frac{a}{2}+2) =$

$O(\log_2 n)$.

Proof:

For every vertex x of G, $f_2(f_1(x)) = a(x+1) + b = ax + a + b = f_1^a(f_2(x))$. Let r be the radius of a section. A section of $(2r+1)$ contiguous vertices on f_1 cycle will broadcast a message along their f_2 edges to $2r+1$ vertices, which are uniformly distributed around the f_1 cycle with the interval of a.

Let $r = \lfloor \frac{a}{2} \rfloor$. When a is even, $r = \frac{a}{2}$. According to the algorithm, an originator will broadcast a message to a section of $2r+1$ contiguous vertices on f_1 cycle in $r+1$ time units. Then at time unit $(r+1)+(r+2)$, a section of $(2r+1)(2r+1) - 2r = (2r)^2 + 2r + 1$ contiguous vertices on f_1 cycle will be informed. Thus, at time unit $(r+1)+(m-1)(r+2)$, a section of $(2r)^m + (2r)^{m-1} + \cdots + 2r + 1 > (2r)^m$ contiguous vertices on f_1 cycle will be informed.

Let m be the least integer such that $(2r)^m \geq n$, i.e., $m = \lceil \log_{2r} n \rceil$.

$$\begin{aligned} b(G) &= (r+1) + (m-1)(r+2) < m(r+2) = \lceil \log_{2r} n \rceil (r+2) \\ &= \lceil \log_a n \rceil (\frac{a}{2} + 2) = \lceil \frac{\log_2 n}{\log_2 a} \rceil (\frac{a}{2} + 2) \end{aligned}$$

When a is odd, $r = \frac{a-1}{2}$. Similarly, an originator will broadcast a message to a section of $2r+1$ contiguous vertices on f_1 cycle in $r+1$ time units. Then at time unit $(r+1)+(r+2)$, a section of $(2r+1)(2r+1)$ contiguous vertices on f_1 cycle will be informed. Thus, at time unit $(r+1)+(m-1)(r+2)$, a section of $(2r+1)^m$ contiguous vertices on f_1 cycle will be informed.

Let m be the least integer such that $(2r+1)^m \geq n$, i.e., $m = \lceil \log_{2r+1} n \rceil$.

$$\begin{aligned} b(G) &= (r+1) + (m-1)(r+2) < m(r+2) = \lceil \log_{2r+1} n \rceil (r+2) \\ &= \lceil \log_a n \rceil (\frac{a-1}{2} + 2) = \lceil \frac{\log_2 n}{\log_2 a} \rceil (\frac{a}{2} + 1.5) \\ &< \lceil \frac{\log_2 n}{\log_2 a} \rceil (\frac{a}{2} + 2) \end{aligned}$$

Therefore, $b(G) < \lceil \frac{\log_2 n}{\log_2 a} \rceil (\frac{a}{2} + 2) = O(\log_2 n)$.

□

The above theorem provides an upper bound equal to $O(\log_2 n)$ on $b(G)$ for one-dimensional graphs G containing $f_1(x) = x + 1$. The broadcasting in G is thus asymptotically optimal. However, the upper bound does not generalize in a simple way to other graphs not containing $f_1(x) = x + 1$.

Broadcasting in unrestricted one-dimensional graphs

We now discuss the broadcasting in another one-dimensional linear congruential congruential graph $G(\{f_1, f_2\}, 2^i)$ where $f_1(x) = 5x + 3$ generates a Hamiltonian cycle, and $f_2(x) = 9x+2$ generates two disjoint cycles of equal length. The following theorem will prove that a section of 2^{i-4} contiguous vertices on f_1 cycle will forward a message along their f_2 edges to 2^{i-4} vertices, which are uniformly distributed around the Hamiltonian cycle with the interval about 16. We thus conjecture that the broadcasting in this graph can be done faster than the previous case.

In this section, we will use $d(x)$ to denote the distance between vertices $f_2(x)$ and $f_2(f_1(x))$ along the f_1 cycle, i.e., $f_2(f_1(x)) = f_1^{d(x)}(f_2(x))$.

THEOREM 4.5.2 *Let $G(\{f_1,\ f_2\},\ 2^i)$ be a one-dimensional linear congruential graph where $f_1(x) = 5x + 3$, $f_2(x) = 9x + 2$, and $i \geq 6$. Let P be a partition on $V(G)$ that consists of the congruence classes modulo 2^{i-6}. For each class $[c]_{2^{i-6}}$, there is a unique k, $0 \leq k < 2^{i-6}$, such that for every vertex x in the class, $d(x) = (4k + 1) \times 16 + 1$. If $x_1,\ x_2 \in V(G)$ are in different classes, then $d(x_1) \neq d(x_2)$. That is, there is a one-to-one mapping between the congruence classes modulo 2^{i-6}, and the set $K = \{k \mid 0 \leq k < 2^{i-6}\}$.*

Proof:

The proof will be done by induction.

Basis: $i = 6$

Since $0 \leq k < 2^{i-6} = 1$, we have to show that the theorem is valid for $k = 0$. That is, for every vertex x of $G(\{f_1, f_2\}, 64)$, $d(x) = (4k + 1) \times 16 + 1 = 17$.

$f_2(f_1(x)) = 9(5x + 3) + 2 = 45x + 29 \bmod 64$

$f_1^{17}(f_2(x)) = f_1^{16}(f_1(f_2(x))) = f_1^{16}(45x + 13) \bmod 64$

$f_1^{16}(x) = 5^{16}x + 3(5^{15} + 5^{14} + \cdots + 1) = 5^{16}x + 3(\frac{5^{16}-1}{4})$

Since $5^{16} \bmod 64 = 1$, and $3(\frac{5^{16}-1}{4}) \bmod 64 = 16$, $f_1^{16}(x) \bmod 64 = x + 16 \bmod 64$.

$f_1^{17}(f_2(x)) = f_1^{16}(45x + 13) = 45x + 29 \bmod 64 = f_2(f_1(x))$

Therefore, the theorem is valid for $i = 6$.

Induction step:

Assume that there is a unique k, $0 \leq k < 2^{i-6}$, for each class $[c]_{2^{i-6}}$ in $V(G(\{f_1, f_2\}, 2^i))$ such that for every vertex x in $[c]_{2^{i-6}}$, $f_2(f_1(x)) = f_1^{d(x)}(f_2(x))$ where $d(x) = 16 \cdot (4k + 1) + 1$.

Let $x = \alpha \cdot 2^{i-6} + c$, and $k' = 16 \cdot (4k + 1)$ for simplicity.

$$f_1^{k'}(x) = 5^{k'}x + 3(5^{k'-1} + 5^{k'-2} + \cdots + 1) = 5^{k'}x + 3(\frac{5^{k'} - 1}{4}) \bmod 2^i$$

Since $5^{16} = 64\beta + 1$, and β is odd, $5^{k'} = 5^{16\cdot(4k+1)} = 5^{64k} \cdot 5^{16} = 64\gamma + 1$ where γ is odd.

$$\begin{aligned}
f_1^{d(x)}(f_2(x)) &= f_1^{16(4k+1)}(f_1(f_2(x))) = f_1^{k'}(f_1(f_2(x))) \\
&= f_1^{k'}(45x + 13) = 5^{k'}(45x + 13) + 3(\frac{5^{k'} - 1}{4}) \\
&= 45 \cdot 5^{k'} \cdot x + 13 \cdot 5^{k'} + 3(\frac{5^{k'} - 1}{4}) \\
&= 45 \cdot 5^{k'} \cdot (\alpha \cdot 2^{i-6} + c) + 13 \cdot 5^{k'} + 3(\frac{5^{k'} - 1}{4}) \\
&= 45 \cdot 5^{k'} \cdot 2^{i-6} \cdot \alpha + 45 \cdot 5^{k'} \cdot c + 13 \cdot 5^{k'} + 3(\frac{5^{k'} - 1}{4}) \\
&= 45 \cdot (64\gamma + 1) \cdot 2^{i-6} \cdot \alpha + 45 \cdot 5^{k'} \cdot c + 13 \cdot 5^{k'} + 3(\frac{5^{k'} - 1}{4}) \\
&= 45 \cdot \gamma \cdot 2^i \cdot \alpha + 45 \cdot 2^{i-6} \cdot \alpha + 45 \cdot 5^{k'} \cdot c + 13 \cdot 5^{k'} + 3(\frac{5^{k'} - 1}{4}) \\
&= 45 \cdot \gamma \cdot 2^i \cdot \alpha + 45 \cdot 2^{i-6} \cdot \alpha + \underbrace{45 \cdot 5^{k'} \cdot c + 13 \cdot 5^{k'} + 3(\frac{5^{k'} - 1}{4})}
\end{aligned}$$

Assume $45 \cdot 5^{k'} \cdot c + 13 \cdot 5^{k'} + 3(\frac{5^{k'}-1}{4}) = \omega_1 \cdot 2^i + t$, $0 \leq t < 2^i$.

$$f_1^{d(x)}(f_2(x)) = 45 \cdot \gamma \cdot 2^i \cdot \alpha + 45 \cdot 2^{i-6} \cdot \alpha + \underbrace{\omega_1 \cdot 2^i + t}$$

where ω_1 depends on c.

Since $f_2(f_1(x)) \bmod 2^i = f_1^{d(x)}(f_2(x)) \bmod 2^i$,

$$\begin{aligned}
f_2(f_1(x)) &= 45x + 29 = 45(\alpha \cdot 2^{i-6} + c) + 29 \\
&= 45 \cdot 2^{i-6} \cdot \alpha + \underbrace{45c + 29}_{} \\
&= 45 \cdot 2^{i-6} \cdot \alpha + \underbrace{\omega_2 \cdot 2^i + t}_{}
\end{aligned}$$

where ω_2 depends on c.

We now consider the extension graph $G(\{f_1, f_2\}, 2^{i+1})$ of the graph $G(\{f_1, f_2\}, 2^i)$. If $\alpha = 2\alpha'$ (i.e., α is even), then $x = \alpha \cdot 2^{i-6} + c = 2\alpha' \cdot 2^{i-6} + c$, i.e., $x \in [c]_{2^{(i+1)-6}}$.

$$\begin{aligned}
f_1^{d(x)}(f_2(x)) \bmod 2^{i+1} &= 45 \cdot \gamma \cdot 2^i \cdot \alpha + 45 \cdot 2^{i-6} \cdot \alpha + \omega_1 \cdot 2^i + t \\
&= 45 \cdot \gamma \cdot 2^i \cdot 2\alpha' + 45 \cdot 2^{i-6} \cdot 2\alpha' + \omega_1 \cdot 2^i + t \\
&= 45 \cdot 2^{i-6} \cdot 2\alpha' + \omega_1 \cdot 2^i + t \pmod{2^{i+1}}
\end{aligned}$$

$$\begin{aligned}
f_2(f_1(x)) \bmod 2^{i+1} &= 45 \cdot 2^{i-6} \cdot \alpha + \omega_2 \cdot 2^i + t \\
&= 45 \cdot 2^{i-6} \cdot 2\alpha' + \omega_2 \cdot 2^i + t \pmod{2^{i+1}}
\end{aligned}$$

If $\alpha = 2\alpha' + 1$ (i.e., α is odd), then $x = \alpha \cdot 2^{i-6} + c = (2\alpha' + 1) \cdot 2^{i-6} + c$, i.e., $x \in [c + 2^{i-6}]_{2^{(i+1)-6}}$.

$$\begin{aligned}
f_1^{d(x)}(f_2(x)) \bmod 2^{i+1} &= 45 \cdot \gamma \cdot 2^i \cdot \alpha + 45 \cdot 2^{i-6} \cdot \alpha + \omega_1 \cdot 2^i + t \\
&= 45 \cdot \gamma \cdot 2^i \cdot (2\alpha' + 1) + 45 \cdot 2^{i-6} \cdot (2\alpha' + 1) + \omega_1 \cdot 2^i + t \\
&= 45 \cdot \gamma \cdot 2^{i+1} \cdot \alpha' + 45 \cdot \gamma \cdot 2^i + 45 \cdot 2^{i-6} \cdot 2\alpha' + 45 \cdot 2^{i-6} \\
&\quad + \omega_1 \cdot 2^i + t \\
&= 2^i + 45 \cdot 2^{i-6} \cdot 2\alpha' + 45 \cdot 2^{i-6} + \omega_1 \cdot 2^i + t \pmod{2^{i+1}} \\
&\quad \text{since } \gamma \text{ is odd.}
\end{aligned}$$

$$\begin{aligned}
f_2(f_1(x)) \bmod 2^{i+1} &= 45 \cdot 2^{i-6} \cdot \alpha + \omega_2 \cdot 2^i + t \\
&= 45 \cdot 2^{i-6} \cdot (2\alpha' + 1) + \omega_2 \cdot 2^i + t \\
&= 45 \cdot 2^{i-6} \cdot 2\alpha' + 45 \cdot 2^{i-6} + \omega_2 \cdot 2^i + t \pmod{2^{i+1}}
\end{aligned}$$

We now consider the following cases based on ω_1 and ω_2, which depend on c.

1. If ω_1 and ω_2 are of the same parity, i.e., they are both even or both odd, then

   (a) when α is even, i.e., $x \in [c]_{2^{i+1-6}}$,

   $$f_2(f_1(x)) \bmod 2^{i+1} = f_1^{d(x)}(f_2(x)) \bmod 2^{i+1}$$

   (b) when α is odd, i.e., $x \in [c + 2^{i-6}]_{2^{i+1-6}}$,

   $$f_2(f_1(x)) \bmod 2^{i+1} = f_1^{d(x)}(f_2(x)) + 2^i \bmod 2^{i+1} = f_1^{d(x)+2^i}(f_2(x)) \bmod 2^{i+1}$$

2. If ω_1 and ω_2 are of different parities, then

   (a) when α is even, i.e., $x \in [c]_{2^{i+1-6}}$,

   $$f_2(f_1(x)) \bmod 2^{i+1} = f_1^{d(x)}(f_2(x)) + 2^i \bmod 2^{i+1} = f_1^{d(x)+2^i}(f_2(x)) \bmod 2^{i+1}$$

   (b) when α is odd, i.e., $x \in [c + 2^{i-6}]_{2^{i+1-6}}$,

   $$f_2(f_1(x)) \bmod 2^{i+1} = f_1^{d(x)}(f_2(x)) \bmod 2^{i+1}$$

Thus each class $[c]_{2^{i-6}}$ in $V(G(\{f_1, f_2\}, 2^{i+1}))$ can be partitioned into two classes $[c]_{2^{i+1-6}}$ and $[c + 2^{i-6}]_{2^{i+1-6}}$. Each of the two classes has a unique $d(x)$ equal to either $16(4k+1)+1$ or $16(4k+1)+1+2^i = 16(4(2^{i-6}+k)+1)+1$ where $0 \leq k < 2^{i-6}$. Therefore, there is a one-to-one mapping between the congruence classes modulo 2^{i+1-6} and the set $K = \{k \mid 0 \leq k < 2^{i+1-6}\}$.

□

For every vertex x of the above graph $G(\{f_1, f_2\}, 2^i)$, $i \geq 6$, the contiguous vertices x, $f_1(x)$, $\cdots$, $f_1^{2^{i-6}-1}(x)$ are each in a unique class $[c]_{2^{i-6}}$, and the vertices $f_1^j(x)$, $f_1^{j+2^{i-6}}(x)$ are in the same class since $f_1(x) = 5x + 3 \bmod 2^{i-6}$ also generates a Hamiltonian cycle. Thus, for each vertex $v = f_1^j(x)$, $0 \leq j < 2^{i-6}$, there is a unique integer k_j, $0 \leq k_j < 2^{i-6}$ such that

$$d(v) = d(f_1^j(x)) = 16(4k_j + 1) + 1 = 16m_j + 1.$$

Therefore, for each vertex x, there is a sequence of integers m_0, m_1, $\cdots$, $m_{2^{i-6}-1}$ such that $4k_j + 1 = m_j$. We will use M_x to denote the sequence $m_0, m_1, \cdots, m_{2^{i-6}-1}$ with respect to x. Actually, every vertex in the same class has the same M_x.

Let the relative positions of the vertices $f_2(x)$, $f_2(f_1(x))$, $\cdots$, $f_2(f_1^j(x))$, $\cdots$, $f_2(f_1^{2^{i-4}}(x))$ on the f_1 cycle be p_0, p_1, $\cdots$, p_j, $\cdots$, $p_{2^{i-4}}$. Then

$$\begin{aligned} p_j &= p_{j-1} + d(f_1^{j-1}(x)) = p_0 + d(x) + d(f_1(x)) + \cdots + d(f_1^{j-1}(x)) \\ &= p_0 + (16m_0 + 1) + (16m_1 + 1) + \cdots + (16m_{j-1\bmod 2^{i-6}} + 1) \\ &= p_0 + 16(m_0 + m_1 + \cdots + m_{j-1\bmod 2^{i-6}}) + j \bmod 2^i \end{aligned}$$

Because $f_1^r(x)$ and $f_1^{r+2^{i-6}}(x)$ are in the same class, $d(f_1^r(x)) = d(f_1^{r+2^{i-6}}(x)) = 16m_r + 1$, $0 \leq r < 2^{i-6}$.

Let $x_0 = 0, x_1, \cdots, x_{2^{i-4}}$ be a sequence of integers generated by M_x in the way that

$$x_j = (x_{j-1} + m_{(j-1\bmod 2^{i-6})}) \bmod 2^{i-4}.$$

Thus $x_j = 0+m_0+m_1+\cdots+m_{j-1\bmod 2^{i-6}} \bmod 2^{i-4}$. Consequently, $p_j = p_0+16x_j+j$. If $x_0, x_1, \cdots, x_{2^{i-4}}$ is a Hamiltonian cycle, then the vertices $f_2(x)$, $f_2(f_1(x))$, $\cdots$, $f_2(f_1^{2^{i-4}-1}(x))$ are almost uniformly distributed.

The following theorem will prove that for every vertex x of G, the sequence of integers x_0, x_1, $\cdots$, $x_{2^{i-4}}$ generated by M_x is a Hamiltonian cycle.

THEOREM 4.5.3 *Let $G(\{f_1, f_2\}, 2^i)$ be a one-dimensional linear congruential graph where $f_1(x) = 5x + 3$, $f_2(x) = 9x + 2$, and $i \geq 6$. For every vertex x of G, if $x_0 = 0$, x_1, $\cdots$, $x_{2^{i-4}}$ is the sequence of integers defined by the sequence $M_x = m_0$, m_1, $\cdots$, $m_{2^{i-6}-1}$ in the way that $x_j = (x_{j-1} + m_{(j-1\bmod 2^{i-6})}) \bmod 2^{i-4}$, then x_0, x_1, $\cdots$, $x_{2^{i-4}}$ is a Hamiltonian cycle.*

Proof:

The proof will be done by induction.

Basis:

- Let $i = 6$, i.e., the size of G is $2^6 = 64$.

  For every vertex x of G, $f_2(f_1(x)) = f_1^{17}(f_2(x))$. $d(x) = 17 = 16m_0 + 1$. Thus $m_0 = 1$. The sequence x_0, x_1, x_2, x_3, x_4 (mod 4) $= 0, 1, 2, 3, 0$ is a Hamiltonian cycle.

- Let $i = 7$, i.e., the size of G is $2^7 = 128$.

  If $x \bmod 2 = 0$, $f_2(f_1(x)) = f_1^{17}(f_2(x))$. $d(x) = 17 = 16 \cdot 1 + 1$.

  If $x \bmod 2 = 1$, $f_2(f_1(x)) = f_1^{81}(f_2(x))$. $d(x) = 81 = 16 \cdot 5 + 1$.

  For every vertex $x \in [0]_2$, $m_0 = 1$, $m_1 = 5$, the sequence $x_0, x_1, x_2, \cdots, x_8 \pmod 8 = 0, 1, 6, 7, 4, 5, 2, 3, 0$ is a Hamiltonian cycle.

  For every vertex $x \in [1]_2$, $m_0 = 5$, $m_1 = 1$, the sequence $x_0, x_1, x_2, \cdots, x_8 \pmod 8 = 0, 5, 6, 3, 4, 1, 2, 7, 0$ is a Hamiltonian cycle.

Induction step:

Let x be a vertex of $G(\{f_1, f_2\}, 2^i)$, and $M_x = m_0, m_1, \cdots, m_{2^{i-6}-1}$. Assume that M_x generates a Hamiltonian cycle $x_0 = 0$, x_1, x_2, $\cdots$, $x_{2^{i-4}} = x_0$. That is, $x_r \neq x_s$ if $r \neq s$.

$$\begin{aligned}
x_{2^{i-6}} &= 0 + m_0 + m_1 + \cdots + m_{2^{i-6}-1} \bmod 2^{i-4} \\
&= (4k_0 + 1) + (4k_1 + 1) + \cdots + (4k_{2^{i-6}-1} + 1) \\
&= 4(k_0 + k_1 + \cdots + k_{2^{i-6}-1}) + 2^{i-6} = 4(0 + 1 + 2 + \cdots + (2^{i-6} - 1)) + 2^{i-6} \\
&\quad (\text{ since } 0 \leq k_j < 2^{i-6}, \text{ and } k_r \neq k_s \text{ if } r \neq s.) \\
&= 4(\frac{1}{2}(2^{i-6})(2^{i-6} - 1)) + 2^{i-6} = 2(2^{i-6})(2^{i-6} - 1)) + 2^{i-6} \\
&= 2(2^{i-6})(2^{i-6}) - 2^{i-6} = (2^{i-4})(2^{i-7}) + (2^{i-4} - 2^{i-6}) \bmod 2^{i-4} \\
&= \frac{3}{4}(2^{i-4})
\end{aligned}$$

Thus $x_{j+2^{i-6}} = x_j + 3 \cdot 2^{i-6}$, $x_{j+2 \cdot 2^{i-6}} = x_j + 2 \cdot 2^{i-6}$, $x_{j+3 \cdot 2^{i-6}} = x_j + 2^{i-6}$, and

$$x_j \bmod 2^{i-6} = x_{j+2^{i-6}} \bmod 2^{i-6} = x_{j+2 \cdot 2^{i-6}} \bmod 2^{i-6} = x_{j+3 \cdot 2^{i-6}} \bmod 2^{i-6}.$$

Therefore, the sequence $x_0, x_1, x_2, \cdots, x_{2^{i-6}} \pmod{2^{i-6}}$ is also a Hamiltonian cycle.

Let $G' = G(\{f_1, f_2\}, 2^{i+1})$ be the extension graph of G, and $x' = x$ or $x + 2^i$ be a vertex of G'.

According to the theorem of edge-change, $f_1^j(x') \bmod 2^{i+1}$ is equal to either $f_1^j(x) \bmod 2^i$ or $(f_1^j(x) \bmod 2^i) + 2^i$. That is, if $(f_1^j(x) \bmod 2^i)$ is in a class $[c]_{2^{i-6}}$, then $(f_1^j(x') \bmod 2^{i+1})$ is also in the class $[c]_{2^{i-6}}$. In addition, since both $f_1(x) = 5x + 3 \bmod 2^{i-6}$ and $f_1(x) = 5x + 3 \bmod 2^{i-5}$ generate a Hamiltonian cycle,

$$f_1^j(x') \bmod 2^{i-6} = f_1^{j+2^{i-6}}(x') \bmod 2^{i-6}$$

$$f_1^j(x') \bmod 2^{i-5} \neq f_1^{j+2^{i-6}}(x') \bmod 2^{i-5}$$

Thus one of the following cases must hold.

$$f_1^j(x') \in [c]_{2^{i-5}}, \text{ and } f_1^{j+2^{i-6}}(x') \in [c+2^{i-6}]_{2^{i-5}} \ldots \text{Case 1.}$$

$$f_1^j(x') \in [c+2^{i-6}]_{2^{i-5}}, \text{ and } f_1^{j+2^{i-6}}(x') \in [c]_{2^{i-5}} \ldots \text{Case 2.}$$

Therefore, according to Theorem 4.5.2, one of the following cases must hold.

$$m'_j = m_j, \text{ and } m'_{j+2^{i-6}} = m_j + 2^{i-4} \ldots \text{Case a.}$$

$$m'_j = m_j + 2^{i-4}, \text{ and } m'_{j+2^{i-6}} = m_j \ldots \text{Case b.}$$

$$\begin{aligned}
x'_{2 \cdot 2^{i-6}} &= 0 + m'_0 + m'_1 + \cdots + m'_{2^{i-6}-1} \\
&\quad + m'_{2^{i-6}} + m'_{2^{i-6}+1} + \cdots + m'_{2 \cdot 2^{i-6}-1} \bmod 2^{i+1-4} \\
&= 2^{i-6} \cdot 2^{i-4} + 2(m_0 + m_1 + \cdots + m_{2^{i-6}-1}) \bmod 2^{i-3} \\
&= 2^{i-7} \cdot 2^{i-3} + 2 \cdot \frac{3}{4} \cdot 2^{i-4} \bmod 2^{i+3} \\
&= \frac{3}{4} \cdot 2^{i-3} \bmod 2^{i-3} = 3 \cdot 2^{i-5}
\end{aligned}$$

That is, $x'_{j+2^{i-5}} = x'_j + 3 \cdot 2^{i-5} \bmod 2^{i-3}$. Therefore,

$$x'_j \neq x'_{j+2^{i-5}} \neq x'_{j+2 \cdot 2^{i-5}} \neq x'_{j+3 \cdot 2^{i-5}}, \text{ and}$$

$$x'_j \bmod 2^{i-5} = x'_{j+2^{i-5}} \bmod 2^{i-5} = x'_{j+2 \cdot 2^{i-5}} \bmod 2^{i-5} = x'_{j+3 \cdot 2^{i-5}} \bmod 2^{i-5}.$$

If $x'_0, x'_1, \cdots, x'_{2^{i-5}}$ (mod 2^{i-5}) is a Hamiltonian cycle, then $x'_0, x'_1, \cdots, x'_{4 \cdot 2^{i-5}}$ (mod 2^{i-3}) is also a Hamiltonian cycle.

For $1 \leq j < 2^{i-6}$, $x'_j \bmod 2^{i-5} = 0 + m'_0 + m'_1 + \cdots + m'_{j-1} \bmod 2^{i-5}$. Because m'_s is equal to either m_s or $m_s + 2^{i-4}$, $x'_j \bmod 2^{i-5} = (\alpha \cdot 2^{i-4}) + (m_0 + m_1 + \cdots + m_{j-1}) \bmod 2^{i-5}$. Because $x_j \bmod 2^{i-6} = m_0 + m_1 + \cdots + m_{j-1} \bmod 2^{i-6}$, $x'_j \bmod 2^{i-5}$ is equal to either $x_j \bmod 2^{i-5}$ or $x_j + 2^{i-6} \bmod 2^{i-5}$.

$$\begin{aligned}
x'_{j+2^{i-6}} \bmod 2^{i-5} &= x'_j + m'_j + m'_{j+1} + \cdots + m'_{2^{i-6}+j-1} \bmod 2^{i-5} \\
&= x'_j + (\beta \cdot 2^{i-4}) + (m_0 + m_1 + \cdots + m_{2^{i-6}-1}) \bmod 2^{i-5} \\
&= x'_j + (\beta \cdot 2^{i-4}) + (2^{i-6} \cdot 2^{i-5} + 2^{i-5} - 2^{i-6}) \bmod 2^{i-5} \\
&= x'_j + 2^{i-6} \bmod 2^{i-5}
\end{aligned}$$

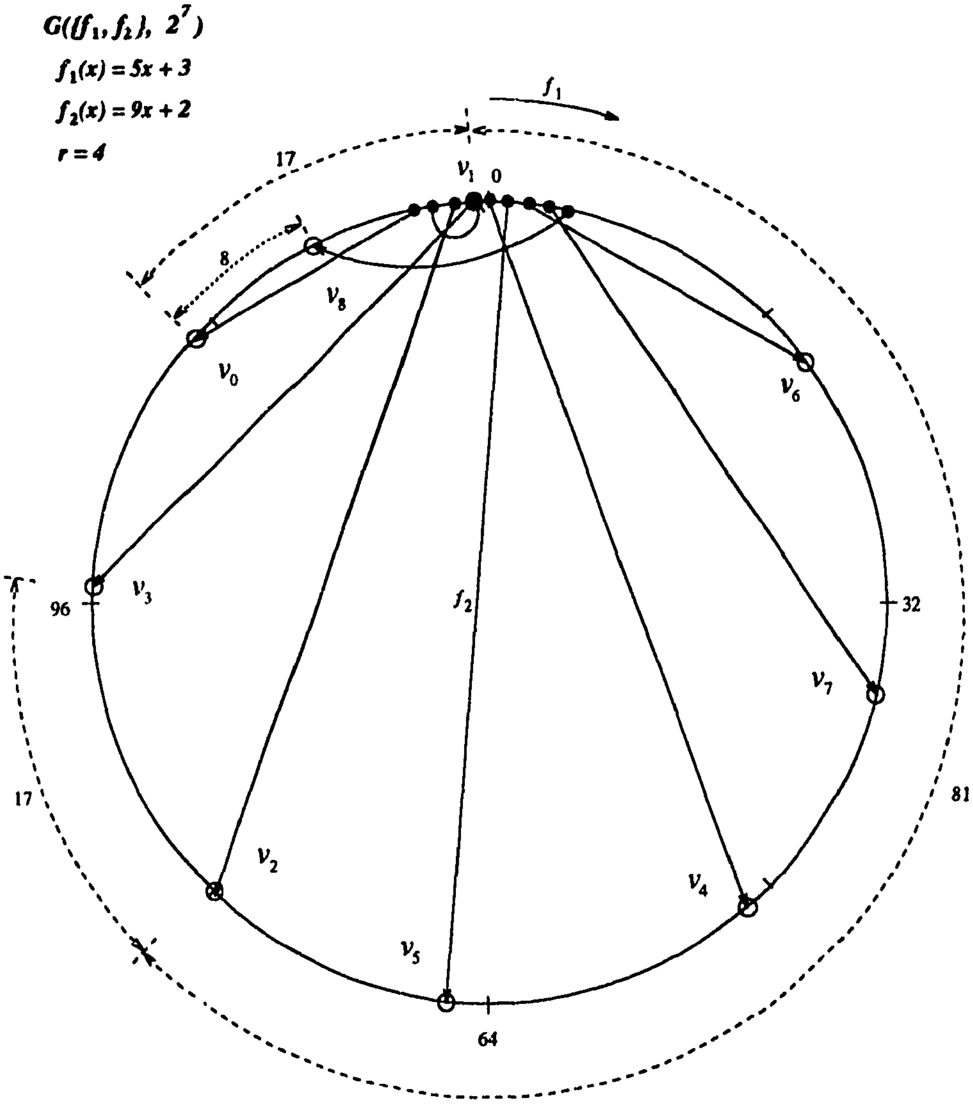


Figure 4.7: Broadcasting in a one-dimensional graph

Because $x_0, x_1, \cdots, x_{2^{i-6}}$ (mod 2^{i-6}) is a Hamiltonian cycle, and because one of the following cases holds,

$$x'_j = x_j \pmod{2^{i-5}}, \text{ and } x'_{j+2^{i-6}} = x_j + 2^{i-6} \pmod{2^{i-5}} \ldots \text{Case 1}$$

$$x'_j = x_j + 2^{i-6} \pmod{2^{i-5}}, \text{ and } x'_{j+2^{i-6}} = x_j \pmod{2^{i-5}} \ldots \text{Case 2}$$

x'_0, x'_1, $\cdots$, $x'_{2^{i-5}}$ (mod 2^{i-5}) is also a Hamiltonian cycle. Consequently, x'_0, x'_1, $\cdots$, $x'_{2^{i+1-4}}$ (mod 2^{i+1-4}) is a Hamiltonian cycle.

□

For example, the sequence of $d(x)$'s is

- 17 if $i = 6$.

- 17, 81 if $i = 7$. (refer to Figure 4.7)

- 145, 81, 17, 209 if $i = 8$..

- 401, 81, 17, 209, 145, 337, 273, 465 if $i = 9$.

We observe that not only the graph $G(\{f_1, f_2\}, 2^i)$ containing $f_1(x) = 5x + 3$ and $f_2(x) = 9x + 2$ has the above properties, but also a graph $G' = G(\{f'_1, f'_2\}, 2^i)$ has the similar properties if $f'_1(x) = 5x + b$, $\gcd(b, 2) = 1$, $f'_2(x) = 9x + d$, $d = 2\alpha$, $\gcd(\alpha, 2) = 1$, and $d \neq 2b$. (If $d = 2b$, G' is vertex-transitive.)

For example, in $G(\{f_1, f_2\}, 2^i)$ where $f_1(x) = 5x + 1$, $f_2(x) = 9x + 10$, the sequence of $d(x)$'s is

- 33 if $i = 6$.

- 33 if $i = 7$.

- 161, 33 if $i = 8$.

- 161, 33, 417, 289 if $i = 9$.

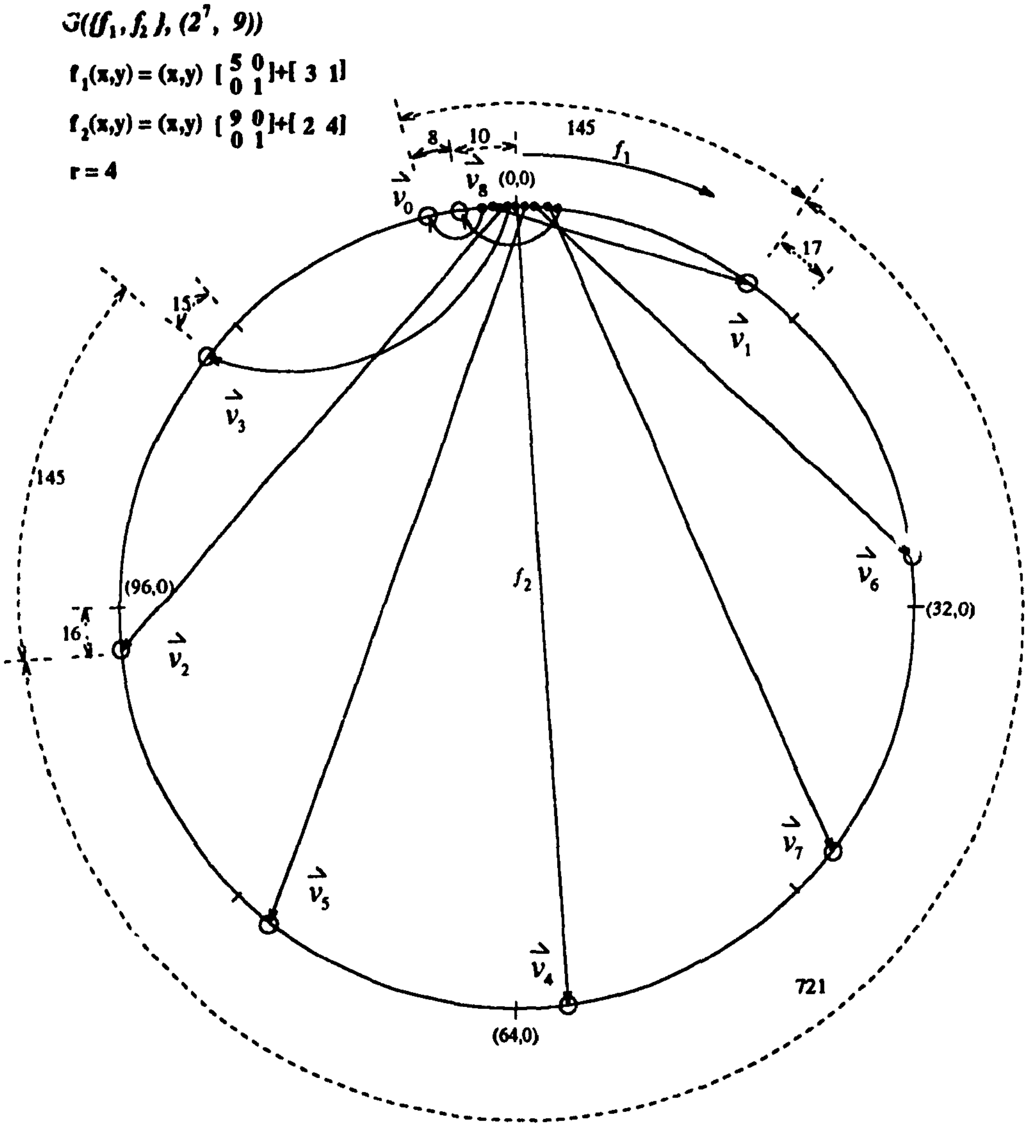
Broadcasting in two-dimensional graphs of simple form

Since a two-dimensional linear congruential graph $G(\{f_1, f_2\}, (2^i, 9))$ of simple form is equal to $G_x(\{f_{1x}, f_{2x}\}, 2^i) \circ G_y(\{f_{1y}, f_{2y}\}, 9)$, the graph G also has similar properties if $f_1(x, y) = (x, y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (b_1, b_2)$, $\gcd(b_1, 2) = 1$, $\gcd(b_2, 9) = 1$, $f_2(x, y) = (x, y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (d_1, d_2)$, $d_1 = 2\alpha$, $\gcd(\alpha, 2) = 1$, $\gcd(d_2, 9) = 1$, and $d_1 \neq 2b_1$. That is, f_1 generates a Hamiltonian cycle, and f_2 generates two disjoint cycles of equal length.

For example, in $G(\{f_1, f_2\}, (2^i, 9))$ where $f_1(x, y) = (x, y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3, 1)$, $f_2(x, y) = (x, y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2, 4)$, the sequence of $d(\vec{v})$'s is

- 145 if $i = 6$.

- 145, 721 if $i = 7$. (refer to Figure 4.8)



$\mathfrak{G}(\{f_1, f_2\}, (2^7, 9))$
$f_1(x,y) = (x,y) \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} + [\,3 \;\; 1\,]$
$f_2(x,y) = (x,y) \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix} + [\,2 \;\; 4\,]$
r = 4
8
10
145
f_1
$\vec{v}_0$
$\vec{v}_8$
(0,0)
17
$\vec{v}_1$
15
$\vec{v}_3$
145
$\vec{v}_6$
(32,0)
(96,0)
16
$\vec{v}_2$
f_2
$\vec{v}_7$
$\vec{v}_5$
$\vec{v}_4$
721
(64,0)

Figure 4.8: Broadcasting in a two-dimensional graph of simple form

- 145, 1873, 1297, 721 if $i = 8$.

- 2449, 4177, 3601, 721, 145, 1873, 1297, 3025 if $i = 9$.

That is, $d(\vec{v}) = (4k+1) \times 16 \times 9 + 1,\ 0 \leq k < 2^{i-6}$.

In the above two-dimensional graphs, a section of 2^{i-4} contiguous vertices on f_1 cycle will forward a message along their f_2 edges to 2^{i-4} vertices, which are uniformly distributed around the Hamiltonian cycle with the interval about 16×9, but not 16.

We have investigated the actual run-time of broadcasting in two-dimensional linear congruential graphs. Table 4.8 shows the *empirical broadcast time*, denoted by $b'(G)$, of graphs $G(\{f_1, f_2\}, (2^i, 9))$ where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2,4)$, and $2 \leq i \leq 13$. We observe that the empirical broadcast time $b'(G)$ of the two-dimensional graph $G(\{f_1, f_2\}, (2^i, 9))$ is much smaller than the upper bound $\lceil \frac{\log_2 n}{\log_2 a} \rceil (\frac{a}{2}+2)$ on broadcast time $b(G_1)$ of the one-dimensional graph $G_1 = G(\{f_1', f_2'\}, n)$ of the same size where $f_1'(x) = x+1$, $f_2'(x) = ax+b$, and $n = 2^i \times 9$.

For example, if $n = 8192 \times 9$, the smallest upper bound on $b(G_1)$ is 32, which is obtained when $a = 5$. However, the empirical broadcast time $b'(G)$ is about 25 when $r \geq 2$. Actually, the algorithm with minor modification can result in a $b'(G)$ less than 25. We will introduce three strategies to speed up the broadcasting later.

Selecting a radius r

How to decide the radius, r, of a section is another issue. Table 4.9 shows the expected broadcast time $(t_1 + t_2)$ where t_1 is the minimum number of time units required to broadcast a message from an originator to all centers of $\frac{n}{2r}$ sections, and t_2 is the number of time units required to broadcast the message from the center of a section to all vertices of the section. t_1 is obtained under the assumption that any two sections have at most an end point in common. As illustrated in the table, when r is small, the minimum number of sections of the Hamiltonian cycle is larger. Thus t_1 is larger. When r is large, although t_1 is thus small, $t_2 = r + 1$ results in a large expected broadcast time $t_1 + t_2$.

The empirical broadcast time b'(G)

Size \ r	1	2	4	8 *	16	32
4 x 9	10	9	9	7		
8 x 9	16	14	13	13	13	
16 x 9	20	18	19	20	20	20
32 x 9	24	19	19	19	19	19
64 x 9	24	19	19	20	20	20
128 x 9	23	19	20	20	20	20
256 x 9	23	19	21	20	20	20
512 x 9	24	21	21	21	21	21
1024 x 9	26	23	22	21	21	21
2048 x 9	27	23	23	22	22	22
4096 x 9	28	25	24	23	23	23
8192 x 9	29	26	24	25	24	24

Table 4.8: The actual run-time of broadcasting

The expected broadcast time (t1 + t2)

r	1		2		4		8 *		16		32	
Size	t1	t2	t1	t2	t1	t2	t1	t2	t1	t2	t1	t2
32 x 9	10	2	9	3	8	5	6	9	5	17	4	33
(t1 + t2)	12		12		13		15		22		37	
64 x 9	11	2	10	3	9	5	8	9	6	17	5	33
	13		13		14		17		23		38	
128 x 9	12	2	11	3	10	5	9	9	8	17	6	33
	14		14		15		18		25		39	
256 x 9	13	2	12	3	11	5	10	9	9	17	8	33
	15		15		16		19		26		41	
512 x 9	15	2	13	3	12	5	11	9	10	17	9	33
	17		16		17		20		27		42	
1024 x 9	16	2	14	3	13	5	12	9	11	17	10	33
	18		17		18		21		28		43	
2048 x 9	17	2	16	3	14	5	13	9	12	17	11	33
	19		19		19		22		29		44	
4096 x 9	18	2	17	3	16	5	14	9	13	17	12	33
	20		20		21		23		30		45	
8192 x 9	20	2	18	3	17	5	16	9	14	17	13	33
	22		21		22		25		31		46	

t1 : minimum number of time units required to broadcast a message to all centers of (n / 2r) sections.

t2 = (r+1) : number of time units required to broadcast a message from the center of a section to all vertices of the section.

Table 4.9: The expected broadcast time

By comparing with the empirical broadcast time $b'(G)$ in Table 4.8, the results are summarized as follows.

1. If $r = 1$, 2 or 4, the empirical broadcast time $b'(G)$ of the graph G is greater than what we expect, $(t_1 + t_2)$.
2. If $r = 16$ or 32, $b'(G)$ is less than $(t_1 + t_2)$.
3. If $r = 8$, $b'(G)$ is close to $(t_1 + t_2)$.

The reason is because the sections are not regularly distributed as we have expected. After an originator broadcasted a message for a period of time, if r is small, there may be some small sections of contiguous vertices on the Hamiltonian cycle that have not been informed yet. To inform the contiguous vertices, at least one of them must receive the message along its f_2 or f_2^{-1} edge. Thus t_1 will be greater than what we expected. On the contrary, if r is large, several vertices of a section may have already received the message along their f_2 or f_2^{-1} edge. The other vertices of the section may in fact be informed earlier than we expected. Thus t_2 will be smaller.

Based on the algorithm, we will now introduce three heuristic strategies to improve the broadcast times of graphs.

Strategy 1

Assume that a vertex is able to receive a message along an edge while it is sending the message along another edge. When a vertex $\vec{v}$ for the first time receives a message, in the next several time units, it will send the message along some edges according to the algorithm. If $\vec{v}$ received the message again along an edge e before $\vec{v}$ sends the message along the edge e, then $\vec{v}$ does not have to send the message along e but will immediately send the message along the next edge.

For example, let e_1, e_2, e_3 and e_4 be the edges incident with $\vec{v}$. Suppose that $\vec{v}$ for the first time receives a message along e_1, and it should send the message sequentially along e_2, e_3, e_4. If $\vec{v}$ receives the message again along e_3 while it is sending the message along e_2. At the next time unit, it sends the message along e_4,

The empirical broadcast time b'(G)

r / Size	1	2	4	8 *	16	32
4 x 9	9	8	8	7		
8 x 9	16	13	12	12	12	
16 x 9	20	17	17	17	17	17
32 x 9	22	17	18	18	18	18
64 x 9	22	18	18	18	18	18
128 x 9	22	18	18	18	18	18
256 x 9	23	19	19	19	19	19
512 x 9	24	21	20	20	20	20
1024 x 9	27	22	21	20	20	20
2048 x 9	27	22	22	21	21	21
4096 x 9	27	24	22	22	22	22
8192 x 9	29	25	23	23	23	23

Table 4.10: The test results of *Strategy* 1

but not e_3, since the vertex adjacent to $\vec{v}$ via e_3 was already informed. Therefore, the vertex adjacent to $\vec{v}$ via e_4 can be informed earlier.

We evaluate the strategy by comparing the empirical broadcast time obtained by the strategy with that obtained by the original algorithm. Table 4.10 shows the empirical broadcast time $b'(G)$ obtained by applying the strategy on the graph G. G is the same as that described in Table 4.8. The new $b'(G)$ is less than the original $b'(G)$ shown in Table 4.8 in general. For example, when $r = 8$, and the size of G is 8192×9, the new $b'(G) = 23$ is 2 less than the original $b'(G)$.

Strategy 2

Let $G_1 = G(F,(2^i,s_2))$ be a two-dimensional linear congruential graph, $G_2 = G(F,(2^{i+1},s_2))$ be the extension of G_1, $b_1 = b((x_0,y_0))$ be the broadcast time of a vertex (x_0,y_0) of G_1, and $b_2 = b((x_0,y_0))$ be the broadcast time of the vertex (x_0,y_0) of G_2. When (x_0,y_0) broadcasts a message in G_2, according to the property of edge change, for every vertex (x,y) of G_2, either (x,y) or $(x+2^i,y)$ or both of them will receive the message after b_1 time units. It seems that the message has been evenly distributed at time unit b_1.

After time unit b_1, if every informed vertex sends the message along its incident

The empirical broadcast time b'(G)

r	1		2		4		8		16		32
Size	(ts)	b'(G)	(ts)	b'(G)	(ts)	b'(G)	(ts)	b'(G)	(ts)	b'(G)	
4 x 9											
8 x 9	(9)	14	(8)	12	(8)	12	(7)	12	(7)	12	
16 x 9	(16)	18	(13)	16	(12)	17	(12)	17	(12)	17	
32 x 9	(20)	21									
64 x 9											
128 x 9											
256 x 9											
512 x 9			(19)	20							
1024 x 9	(24)	25									
2048 x 9											
4096 x 9			(22)	23							
8192 x 9	(27)	28									

Note : Strategy 2 is executed after time stamp ts.

Table 4.11: The test results of *Strategy* 2

edges that the message has not traversed along, b_2 is hopefully one greater than b_1.

In the strategy, a *time stamp* and a *clock* will be appended to a message. The time stamp is initialized to be zero by the originator. Whenever a vertex wants to send the message, it increases the time stamp by one. The clock is used to tell vertices when the strategy should be executed.

For each case in Table 4.10, if $b'(G_2) > b'(G_1) + 1$, we apply *Strategy 2* on the graph G_2 after time $b'(G_1)$. Table 4.11 shows the results. If the graph G_2 is of size 8×9 or 16×9, $b'(G_2)$ is still much greater than $b'(G_1)$, else $b'(G_2) = b'(G_1) + 1$.

The reason why $b'(G_2)$ is much greater than $b'(G_1)$ for the graph G_2 of size 8×9 or 16×9 is because G_2 is vertex-transitive. That is, there exists an integer n such that $f_2(x, y) = f_1^n(x, y)$ for every vertex of G_2. Any two adjacent vertices on the Hamiltonian cycle send a message along their f_2 edges always to two adjacent vertices. Thus the message cannot be rapidly broadcasted.

Strategy 3

In the original algorithm, when the center of a section receives a message, the message can be sent to a vertex of other section by some vertices of the section three

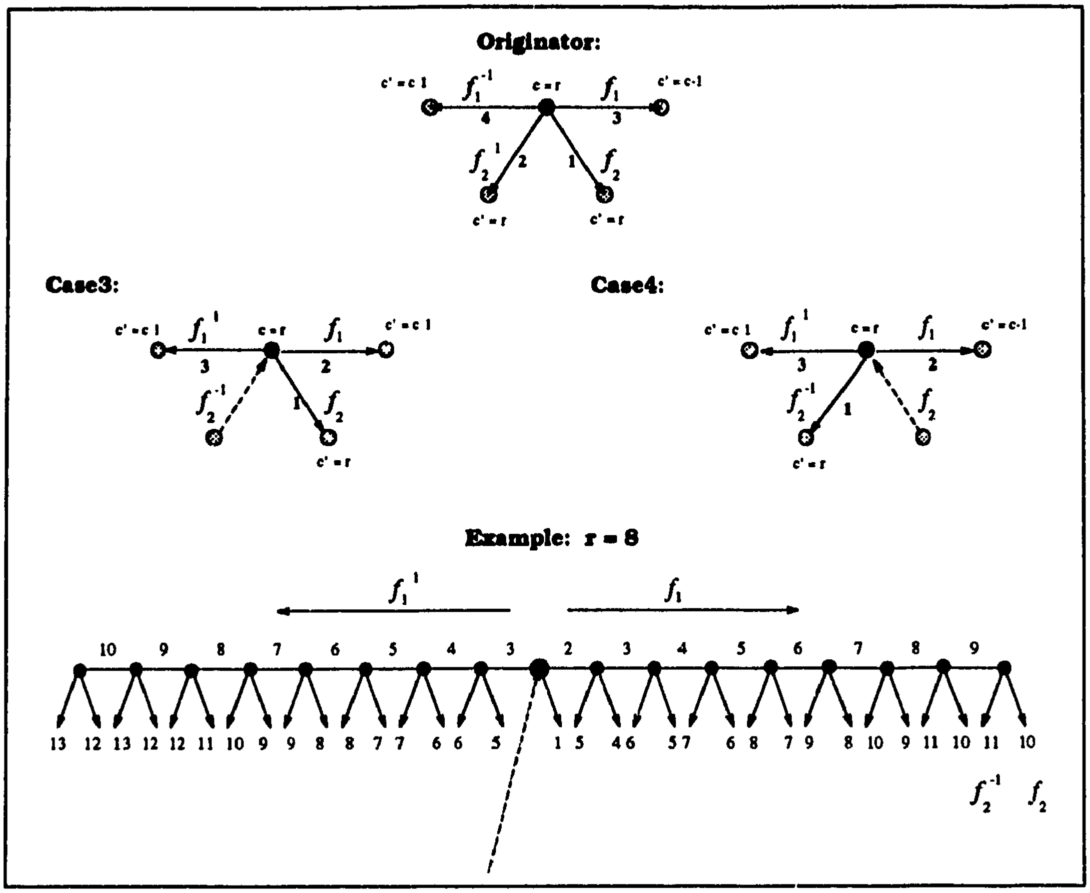


Figure 4.9: Modified broadcasting algorithm for *Strategy* 3

units of time later, and then at least two vertices will send the message to the vertices of other sections per unit of time.

Since the graph of simple form of size 4 × 9, 8 × 9 or 16 × 9 are vertex-transitive, any two adjacent vertices on the Hamiltonian cycle send message along their f_2 or f_2^{-1} edges always to two adjacent vertices. The result is similar to when only one of them sends the message along its f_2 or f_2^{-1} edge. Thus we only have to consider how early the first vertex of a section will send the message along its f_2 or f_2^{-1} edge. In other words, if the center sends the message along its f_2 or f_2^{-1} edge as soon as possible, t_1 can be smaller.

Figure 4.9 illustrates the modified algorithm. Only the order of sending message is changed at the originator and the vertices that receives message along its f_2 or f_2^{-1} edge (i.e., the center of a section). For each section, the center sends message to another section immediately after it received the message, and t_2 is only 1 greater than that of the original algorithm.

Comparison of empirical broadcast times

Size	b'(G)	b"(G)
4 x 9	7	7
8 x 9	12	11
16 x 9	17	12
32 x 9	18	14
64 x 9	18	15
128 x 9	18	15
256 x 9	19	16
512 x 9	20	18
1024 x 9	20	19
2048 x 9	21	20
4096 x 9	22	22
8192 x 9	23	23

Table 4.12: Evaluation of *Strategy* 3

Table 4.12 shows the empirical broadcast time of the graph $G(\{f_1, f_2\}, (2^i, 9))$ where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} + (3,1)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix} + (2,4)$, and $2 \leq i \leq 13$ with radius $r = 8$. We use $b'(G)$ to denote the empirical broadcast time obtained by applying *Strategy* 1, and $b''(G)$ to denote the empirical broadcast time obtained by applying both *Strategy* 1 and *Strategy* 3. As shown in Table 4.12, $b''(G)$ is less than $b'(G)$ for $4 \leq i \leq 12$.

4.5.3 Evaluation

As shown in Table 4.8 and 4.9, when the radius $r = 8$, the empirical broadcast time $b'(G)$ is close to the expected broadcast time $t_1 + t_2$. Thus the broadcasting algorithm is evaluated with $r = 8$, and the *Strategy* 1 is used to obtain smaller $b'(G)$. We investigate the empirical broadcast time of two-dimensional linear congruential graphs $G(\{f_1, f_2\}, (2^i, 9))$ where $f_1(x,y) = (x,y)\begin{pmatrix} 5 & 0 \\ a_{21} & 1 \end{pmatrix} + (b_1, 1)$, $f_2(x,y) = (x,y)\begin{pmatrix} 9 & 0 \\ c_{21} & 1 \end{pmatrix} + (d_1, d_2)$, b_1 satisfies lemma 3.2.1 (f_1 generates a Hamiltonian cycle), d_1 satisfies lemma 3.2.2, $\gcd(d_2, 9) = 1$, and $2 \leq i \leq 13$.

For graphs of simple form ($a_{21} = c_{21} = 0$), we choose $b_1 \in \{1, 3, 5, 7, 9, 11, 13, 15\}$, $d_1 \in \{2, 6, 10, 14\}$, and $d_2 \in \{4, 5, 7, 8\}$. Thus f_2 generates two disjoint cycles of equal length. Some graphs with generators described as above have large diameter. For example, if $b_1 = 1$, $d_1 = 14$, $d_2 = 4$, and $i = 4$, the diameter of G is greater than 15. We only choose the graphs whose diameters are about 12 when $i = 10$. The graphs with following parameters satisfy the condition.

1. $d_2 = 4$ or 7, and $(b_1, d_1) \in \{(1,6), (3,2), (5,14), (7,10), (9,6), (11,2), (13,14), (15,10)\}$.
2. $d_2 = 5$ or 8, and $(b_1, d_1) \in \{(1,14), (3,10), (5,6), (7,2), (9,14), (11,10), (13,6), (15,2)\}$.

We can partition the above graphs into four sets of graphs according to d_2, because we observed that any two graphs in the same set have similar broadcast times. In the set of graphs with $d_2 = 7$, every graph G of size $4 \times 9 = 36$ has $b'(G) = 18$, because $f_2(x,y) = f_1^{34}(x,y)$ for every vertex (x,y) of G. Thus, in the following comparison with de Bruijn graphs, Kautz graphs and 5-star graph, this set of graphs is not considered.

For graphs of complex form, we choose $a_{21} \in \{2,3\}$, $c_{21} \in \{2,3,4,6,7,8\}$, and $(d_1,d_2) \in \{(6,4),(14,5),(6,7),(14,8)\}$. The graphs with the above parameters are of diameter 10 or 11 when $i = 10$.

Table 4.13 lists the maximum $b'(G)$ and minimum $b'(G)$ over the above graphs of simple form and complex form. We will use G_s to denote a graph of simple form, and G_c to denote a graph of complex form. As a result of the investigation, the empirical broadcast time $b'(G_s)$ is greater than $b'(G_c)$ if G_s and G_c are of the same size. The reason is because G_s is more symmetric than G_c.

The table also shows the upper bound on broadcast time $\frac{d+1}{2}D + \frac{d}{2}$ of undirected de Bruijn graph $UB(d,D)$ of size d^D where $d = 2$, $5 \leq D \leq 16$, and the upper bound on broadcast time $\frac{d+1}{2}D + \frac{2d-1}{2}$ of undirected Kautz graph $UK(d,D)$ of size $(d+1)(d^{D-1})$ where $d = 2$, $4 \leq D \leq 15$.

For graphs of similar sizes and of the same degree, their broadcast times are compared. If $d = 2$ and $D = i + 3$, the size of $G(F,(2^i,9))$ is greater than the size of $UB(d,D)$, if $d = 2$ and $D = i+2$, the size of $G(F,(2^i,9))$ is greater than the size of $UK(d,D)$, and if $i = 4$, the size of $G(F,(2^i,9))$ is greater than the size of 5-star. We summarize the comparison as follows:

UB(2,D) Size= 2^D		UK(2,D) Size= $3 * 2^{D-1}$		G({f1, f2}, (2^i, 9)) Size= $2^i \bullet 9$				
					Simple Form		Complex Form	
D	*b(UB(2,D))*	*D*	*b(UK(2,D))*	*i*	*Min(b'(Gs))*	*Max(b'(Gs))*	*Min(b'(Gc))*	*Max(b'(Gc))*
5	9	4	7	2	7	9	7	10
6	10	5	9	3	11	15	8	12
7	12	6	11	4	15	18	10	12
8	13	7	12	5	16	18	11	13
9	15	8	14	6	16	18	13	14
10	16	9	15	7	17	19	14	15
11	18	10	17	8	18	20	15	16
12	19	11	18	9	19	20	17	18
13	21	12	20	10	19	21	18	19
14	22	13	21	11	21	22	19	21
15	24	14	22	12	22	23	20	22
16	25	15	24	13	23	24	22	23

Table 4.13: Evaluation of empirical broadcast time

- For graphs $G_c = G(\{f_1, f_2\}, (2^i, 9))$ of complex form,

  1. if $2 \leq i \leq 13$, $\min(b'(G_c)) < b(UB(2, i+3))$, and $\min(b'(G_c)) < b(UK(2, i+2))$;
  2. if $4 \leq i \leq 13$, $\max(b'(G_c)) \leq b(UB(2, i+3))$;
  3. if $6 \leq i \leq 13$, $\max(b'(G_c)) \leq b(UK(2, i+2))$;
  4. if $i = 4$, $b'(G_c) \leq b(5\text{-star}) = 12$.

- For graphs $G_s = G(\{f_1, f_2\}, (2^i, 9))$ of simple form,

  1. if $8 \leq i \leq 13$, $\min(b'(G_s)) \leq b(UB(2, i+3))$;
  2. if $10 \leq i \leq 13$, $\min(b'(G_s)) \leq b(UK(2, i+2))$;
  3. if $10 \leq i \leq 13$, $\max(b'(G_s)) \leq b(UB(2, i+3))$;
  4. if $i = 13$, $\max(b'(G_s)) = b(UK(2, i+2))$;
  5. if $i = 4$, $b'(G_s) > b(5\text{-star})$.

In general, $b'(G_c)$ and $b'(G_s)$ are similar to $b(UB(2, i+3))$ and $b(UK(2, i+2))$ with the exception of $b'(G_s)$ for $3 \leq i \leq 8$, which is greater than $b(UB(2, i+3))$ and $b(UK(2, i+2))$. However, we can use *Strategy* 3 to reduce $b'(G_s)$. We conjecture that the new $b'(G_s)$ will be close to $b''(G)$ shown in Table 4.12, which is similar to $b(UB(2, i+3))$ and $b(UK(2, i+2))$.

Therefore, we conclude that broadcasting in a two-dimensional linear congruential graph can be done in time, which is very similar to the time for broadcasting in de Bruijn and Kautz graphs of similar sizes.

Chapter 5

Conclusion

5.1 Research Results

The multicomputers and distributed systems are designed to improve performance, resource utilization, and to provide users with a more convenient and reliable environment. In the design of these systems, the network topology as well as routing and broadcasting algorithms are very important issues. The two-dimensional linear congruential graphs have been previously shown to have many desirable properties of interconnection networks such as large size, small degree, small diameter, regularity and maximal connectivity. We were interested in the problems for routing and broadcasting in two-dimensional linear congruential graphs. Our research was focused on the graphs $G(\{f_1, f_2\}, (2^i, s_2))$ of degree 4 where f_1 generates a Hamiltonian cycle, and f_2 generates a small number of disjoint cycles.

In general, two-dimensional linear congruential graphs are not vertex-transitive. However, they have specific symmetric properties. Depending on the generators, two-dimensional linear congruential graphs can be partitioned into two classes: graphs of simple form and graphs of complex form. We have provided sufficient conditions for a graph of simple form to be vertex-transitive with respect to a partition. The partition on the values of x coordinate of all vertices consists of congruence classes modulo 2^{i-4}. We have also provided sufficient conditions for a graph of complex form to be quarter symmetric. Based on their symmetric properties, routing and broadcasting algorithms were proposed.

The global routing algorithm uses a breadth first search approach. For a graph of size $2^i \times 9$, the number of path tables required is 2^{i-4} if the graph is of simple form, or $2^{i-2} \times 9$ if the graph is of complex form. As the size of graph increases, the number of path tables is very large.

A more efficient global routing algorithm for graphs of simple form was thus proposed. In the scheme, only a one-to-one mapping path table and a size-independent (16×9 memory entries) mapping table are required. The paths determined by this scheme may not be the shortest. However, the maximum length of any path is very close to the diameter.

We proposed two distributed routing algorithms suitable for different faulty conditions. They both use a depth first search approach and can route messages along the shortest path if no faults are encountered. The two algorithms have been evaluated under various faulty conditions. The empirical results reveal that in presence of faults, the progressive algorithm may route messages along a shorter path than that routed by the conservative algorithm if the number of faults is small. However, the progressive algorithm does not guarantee a path for two connected vertices in specific cases of faults.

The algorithm of finding disjoint paths for a pair of vertices recursively executes a procedure until four disjoint paths between them are found. The set of disjoint paths may not contain the shortest paths. However, their lengths are bounded by the level n of recursive calls to the procedure, i.e., $len(p) \leq min\_len + 4 + 2n$ where min_len is the distance between the pair of vertices. In other words, the difference between lengths of any two disjoint paths in the set is at most $4 + 2n$. We observed that the maximum n for any two vertices of a graph is small.

The broadcasting algorithm provides an upper bound $O(\log_2 n)$ on broadcast time of any one-dimensional linear congruential graph containing generator $f_1(x) = x + 1$. The broadcast time is asymptotically optimal. However, it does not generalize in a simple way to two-dimensional graphs. We observed that, with a proper choice of generators, the broadcast time of a two-dimensional graph is smaller than the upper bound and is similar to the time required for broadcasting in the de Bruijn graph of the same degree and comparable size.

5.2 Future Considerations

Some problems are worth of further study and we mention some of them below.

- Because the structure of a graph of complex form is more sophisticated than that of simple form, the global routing algorithm with a unique path table cannot be applied on a graph of complex form. Thus the global routing in a graph of complex form still has to create large number of tables.

- In the problem of distributed routing, the maximum number of message hops required for a detour due to a faulty link is not known yet.

- The broadcasting in two-dimensional linear congruential graphs can be done in time similar to the time required for de Bruijn graphs. However, no good upper bound on the broadcast time of a two-dimensional linear congruential graph has been found yet.

Bibliography

[1] C. C. Koung. Multidimensional Linear Congruential Graphs: A New Model for Large-Scale Interconnection Networks. Master's thesis, Dept. of Comp. Sci. Concordia University, Montreal, Quebec, Canada, 1993.

[2] F.R.K. Chung. Diameters of Graphs: Old Problems and New Results. In *Proceedings of the 18th Southeastern Conference on Combinatorics, Graph Theory and Computing Congr. Numer.*, pages 295–317, 1987.

[3] J. C. Bermond and C. Delorme. Strategies for Interconnection Networks: Some Methos from Graph Theory. *Journal of Parallel and Distributed Computing*, 3:433–449, 1986.

[4] C. L. Seitz. The Cosmic Cube. *Commun. ACM*, 28(1):22–33, Jan. 1985.

[5] Y. Saad and M. H. Schultz. Topological Properties of Hypercubes. *IEEE Transactions on Computers*, 37(7):867–872, Jul. 1988.

[6] B. Becker and H. U. Simon. How Robust is the n-Cube? In *IEEE Annual Symposium on Fundations of Computer Science*, pages 283–291, 1986.

[7] J. C. Bermond and C. Peyrat. de Bruijn and Kautz network: a competitor for the hypercube ? In F. Andre and J.P. Verjus, editors, *Proceedings of the first European Workshop on Hypercubes and Distributed Computers*, pages 279–293, (North-Holland), 1989. Elsevier Science Publishers B.V.

[8] M. R. Samatham and D. K. Pradhan. The De Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI. *IEEE Transactions on Computers*, 38(4):567–581, Apr. 1989.

[9] M. R. Samatham and D. K. Pradhan. Correction to The De Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI. *IEEE Transactions on Computers*, 40(1):122, 1991.

[10] S. B. Akers and B. Krishnamurthy. A Group-Theoretic Model for Symmetric Interconnection Networks. *IEEE Transactions on Computers*, 38(4):555–566, Apr. 1986.

[11] S. B. Akers, D. Harel, and B. Krishnamurthy. The Star Graph: An Attractive Alternative to the n-cube. In *International Conference on Parallel Processing*, pages 393–400, 1987.

[12] S. B. Akers and B. Krishnamurthy. The Fault Tolerance of Star Graphs. In *Proceedings of the 2nd International Conference on Supercomputing*, pages 270–279, 1987.

[13] J. Opatrny, D. Sotteau, N. Srinivasan, and K. Thulasiraman. DCC Linear Congruential Graphs: A New Class of Interconnection Network. *IEEE Transactions on Computers.*

[14] P. Fraigniaud. Asymptotically Optimal Broadcasting and Gossiping in Faulty Hypercube Multicomputers. *IEEE Transactions on Computers*, 41(11):1410–1419, Nov. 1992.

[15] K. H. Rosen. *Discrete Mathematics and its Applications.* Random House, 1988.

[16] M. N. S. Swamy and K. Thulasiraman. *Graphs, Networks, and Algorithms.* John Wiley and Sons, Inc., 1981.

[17] K. A. Ross and C. R. B. Wright. *Discrete Mathematics.* Prentice-Hall, Inc., second edition, 1988.

[18] W. D. Hillis. *The Connection Machine.* The MIT Press, 1985. An ACM Distinguished Dissertation.

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

[20] J. P. Hayes, T. N. Mudge, and Q. F. Stout. Architecture of a Hypercube Supercomputer. In *Proceedings of International Conference on Parallel Processing*, pages 653-660, Aug. 1986.

[21] M. S. Chen and K. G. Shin. Adaptive Fault-Tolerant Routing in Hypercube Multicomputers. *IEEE Transactions on Computers*, 39(12):1406–1416, Dec. 1990.

[22] T. C. Lee and J. P. Hayes. A Fault-Tolerant Communication Scheme for Hypercube Computers. *IEEE Transactions on Computers*, 41(10):1242–1256, Oct. 1992.

[23] M. S. Chen and K. G. Shin. Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Computers. *IEEE Transactions on Parallel Distributed System*, 1(2):152-159, Apr. 1990.

[24] N. G. de Bruijn. A Combinatorial Problem. *Koninklje Nedderlandse Academie van Wetenshappen Proc.*, A 49:758-764, 1946.

[25] D. K. Pradhan and S. M. Reddy. A Fault-Tolerant Communication Architecture for Distributed Systems. *IEEE Transactions on Computers*, c-31(9):863-870, Sep. 1982.

[26] A. H. Esfahanian and S. L. Hakimi. Fault-Tolerant Routing in DeBruijn Communication Network. *IEEE Transactions on Computers*, c-34(9):152–159, Sep. 1985.

[27] Z. Liu. Optimal Routing in the De Bruijn Networks. In *Proceedings of the Tenth International Conference on Distributed Computing System*, pages 537-544. IEEE Comput. Soc. Press, 1990.

[28] J. C. Bermond and C. Peyrat. Broadcasting in de Bruijn Networks. In *Proceedings of the 19th Southeastern Conference on Combinatorics, Graph Theory and Computing Congr. Numer. 66*, pages 292–283, 1988.

[29] M. C. Heydemann, J. Opatrny, and D. Sotteau. Broadcasting and spanning trees in de Bruijn and Kautz networks. *Discrete Applied Mathematics*, 37/38:297-317, 1992.

[30] V. E. Mendia and D. Sarkar. Optimal Broadcasting on the Star Graph. *IEEE Transactions on Parallel and Distributed System*, 3(4), Jul. 1992.

[31] S. B. Akers and B. Krishnamurthy. Group Graph as International Network. In *Proceedings of 14th International Conference on Fault Tolerant Computing*, pages 422-427, 1984.

[32] N. Bagherzadeh, N. Nassif, and S. Latifi. *A Routing and Broadcasting Scheme on Faulty Star Graph.* Department of Electrical and Computer Engineering, University of California, Irvine, e-mail: nader@balboa.eng.uci.edu.

[33] A. L. Liestman and J. G. Peters. Broadcast Network of Bounded Degree. *Siam J. of Disc. Math.*, 1:531-540, 1988.