INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
INTERCONNECTION NETWORKS WITH PERMUTATION FUNCTIONS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By
I-pieng Kao, B.S.E.E., M.S.

* * * * *

The Ohio State University
1984

Reading Committee:
Tse-yun Feng, Chairman
G. Neil Robertson
Jerome Rothstein
Kyungsook Y. Lee

Approved By

Adviser
Department of Computer
and Information Science
©
Copyright by
I-pieng Kao
1983
To My Parents
ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my advisor, Professor Tse-yun Feng, for initiating this research, for discussing the material, and for making superb suggestions throughout the accomplishment of this dissertation. I would also like to thank him for the advisement in scholarly, research, and professional activities during my doctoral study.

I wish to thank Professor G. Neil Robertson for mathematical guidance, valuable discussions, and his friendship. Thanks to Professor Jerome Rothstein and Dr. Paul D. Seymour for stimulating ideas and to Professor Kyungsook Y. Lee for helpful comments.

Thanks are also due to the IEEE Computer Society for the Richard E. Merwin Scholarship and to the Instruction and Research Computer Center for the financial support as a graduate research associate along the way of this study.

There are many people who have helped, especially Professor Dennis W. Leinbaugh, Professor Venkataraman Ashok, Professor Bruce W. Weide, Professor Ming-Tsan Liu, Professor William F. Ogden, Mr. Mark J. Watkins, and Mr. Phillip R. Saksa, to name a few. To all these people, I express my hearty appreciation.

Finally, I would like to thank my loving wife, Lin-jin, for her endless support, encouragement and understanding throughout the entire period of my graduate study.
VITA

November 26, 1955 . . Born - Taipei, Taiwan, R. O. C.

1977 . . . . . . B.S.E.E., National Taiwan University, Taipei, Taiwan, R. O. C.

1977-1979 . . . . Officer, Military Police, R. O. C.

1979-1980 . . . . Teaching Associate, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio

1980 . . . . . . M.S., The Ohio State University, Columbus, Ohio

1981-1983 . . . . Research Associate, Instruction and Research Computer Center, The Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in Computer Architecture.
Professor Tse-yun Feng

Studies in Computer Operating Systems.
Professor Dennis W. Leinbaugh

Studies in Algorithm Design and Analysis.
Professor Venkataraman Ashok
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DEDICATION</td>
<td>ii</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>VITA</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1. General Background</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2. Definitions</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.2.1. Interconnection Networks</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.2.2. Binary Element Multistage Interconnection Networks</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>1.2.3. Permutation Networks</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>1.3. A Brief Survey</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>1.4. The Objective of Investigation</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>2. An Algorithmic Approach to Dynamic Interconnection Networks</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>2.1. The Particular Permutation Realization (PPR) Problem</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>2.2. Some Polynomial Solutions</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>2.3. Problems Related to PPR</td>
<td>39</td>
</tr>
</tbody>
</table>
2.3.1. Problems Equivalent to PPR

2.3.2. The Vertex-Version of PPR (VPPR) Problem

2.3.3. More Related Problems

2.4. The VPPR Problem is NP-Complete

2.5. The NP-Completeness of PPR

2.6. Parameters Tau and Nu in Graphs and Hypergraphs

2.6.1. Generalization to Hypergraphs

2.7. Summary

3. Efficient Membership-Recognition and Permutation-Recognition Classes of Multistage Interconnection Networks

3.1. Classifying Multistage Interconnection Networks by the Particular Permutation Realization Problem

3.2. The Reverse Network

3.3. Topological Equivalence

3.4. Transversal Networks and Partial Transversal Networks

3.4.1. The General and the Standard Benes Binary Network

3.4.2. Partial Transversal Networks

3.5. Cutting Transversal Networks in the Middle

3.5.1. Conjugate Networks

3.5.2. Bit-Controlled Networks with Single Outlet-Address

3.6. Beyond Partial Transversal Networks and their Reverse Networks

3.6.1. Minimum Full Access Networks

3.6.2. Rearrangeability Considerations
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1. Number of Permutations of PTNs of Size 8</td>
<td>102</td>
</tr>
<tr>
<td>3.2. Permutation Number of Partial (Omega)2-1 of Size 8</td>
<td>128</td>
</tr>
<tr>
<td>5.1. A Binary Element Tested for Single Link Faults</td>
<td>186</td>
</tr>
<tr>
<td>5.2. A Binary Element Tested for Single Control Faults</td>
<td>187</td>
</tr>
<tr>
<td>5.3. The 16 Possible States of a Binary Element</td>
<td>189</td>
</tr>
<tr>
<td>5.4. A Binary Element Tested for s10-Malfunctions</td>
<td>190</td>
</tr>
<tr>
<td>5.5. A Binary Element Tested for s5-Malfunctions</td>
<td>191</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Interconnection Scheme of ILLIAC IV</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Outline of Burroughs Scientific Processor</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>General Scheme of Associative Processors</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>Interconnection Schemes for Multiprocessors</td>
<td>9</td>
</tr>
<tr>
<td>1.5</td>
<td>Interconnection Network</td>
<td>12</td>
</tr>
<tr>
<td>1.6</td>
<td>One-Way Two-Sided Interconnection Network</td>
<td>12</td>
</tr>
<tr>
<td>1.7</td>
<td>Two States of a Binary Element</td>
<td>12</td>
</tr>
<tr>
<td>1.8</td>
<td>The n*m BEMIN</td>
<td>16</td>
</tr>
<tr>
<td>1.9</td>
<td>Benes Binary Network of Size 8</td>
<td>17</td>
</tr>
<tr>
<td>1.10</td>
<td>A Full Switch</td>
<td>23</td>
</tr>
<tr>
<td>1.11</td>
<td>A Partitioning Network</td>
<td>23</td>
</tr>
<tr>
<td>1.12</td>
<td>A Generalized Connection Network</td>
<td>24</td>
</tr>
<tr>
<td>1.13</td>
<td>Two Extended States for a Binary Element</td>
<td>24</td>
</tr>
<tr>
<td>1.14</td>
<td>Binary Element with Comparison Capability</td>
<td>24</td>
</tr>
<tr>
<td>2.1</td>
<td>The 2-Stage Case</td>
<td>36</td>
</tr>
<tr>
<td>2.2</td>
<td>The 3-Stage Case</td>
<td>36</td>
</tr>
<tr>
<td>2.3</td>
<td>A Counterexample</td>
<td>38</td>
</tr>
<tr>
<td>2.4</td>
<td>The 4-Vertex Digraph</td>
<td>42</td>
</tr>
<tr>
<td>2.5</td>
<td>An Example of Contraction</td>
<td>42</td>
</tr>
<tr>
<td>2.6</td>
<td>An Example of BEMIN and VBMIN Cyclic Digraphs</td>
<td>43</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.10</td>
<td>Small Rearrangeable Networks based on LRTs</td>
<td>127</td>
</tr>
<tr>
<td>3.11</td>
<td>Small Rearrangeable Networks based on LTs &amp; RTs</td>
<td>130</td>
</tr>
<tr>
<td>4.1</td>
<td>Circulating Network</td>
<td>139</td>
</tr>
<tr>
<td>4.2</td>
<td>Buffered Extended Circulating Network</td>
<td>142</td>
</tr>
<tr>
<td>4.3</td>
<td>The BSN of Single-Stage Circulating Networks</td>
<td>152</td>
</tr>
<tr>
<td>4.4</td>
<td>Ring Topology</td>
<td>157</td>
</tr>
<tr>
<td>4.5</td>
<td>Some Non-PCN Strongly-Connected Networks</td>
<td>159</td>
</tr>
<tr>
<td>4.6</td>
<td>A BSN for a Permutation Circulating Network</td>
<td>166</td>
</tr>
<tr>
<td>4.7</td>
<td>An Example for Clusters and Bundles</td>
<td>170</td>
</tr>
<tr>
<td>4.8</td>
<td>Tree Network of Size 16</td>
<td>173</td>
</tr>
<tr>
<td>4.9</td>
<td>An Implementation of Shuffle, Cyclic-Shift Network</td>
<td>176</td>
</tr>
<tr>
<td>4.10</td>
<td>An Outshuffle, Inshuffle Network</td>
<td>178</td>
</tr>
<tr>
<td>4.11</td>
<td>A Permutation not Realizable</td>
<td>180</td>
</tr>
<tr>
<td>4.12</td>
<td>An Unshuffle Solution</td>
<td>180</td>
</tr>
<tr>
<td>5.1</td>
<td>Fault-Free Response Scheme for Single Link Faults</td>
<td>197</td>
</tr>
<tr>
<td>5.2</td>
<td>An Alternate Test Scheme</td>
<td>198</td>
</tr>
<tr>
<td>5.3</td>
<td>An Example of Locating the Link Fault</td>
<td>200</td>
</tr>
<tr>
<td>5.4</td>
<td>Diagnosis Case 1 of Single Element Logical Faults</td>
<td>207</td>
</tr>
<tr>
<td>5.5</td>
<td>Diagnosis Case 2 of Single Element Logical Faults</td>
<td>211</td>
</tr>
<tr>
<td>5.6</td>
<td>The Existence of '—' Faults</td>
<td>215</td>
</tr>
<tr>
<td>5.7</td>
<td>Locating Single Link Faults Asynchronously</td>
<td>220</td>
</tr>
<tr>
<td>5.8</td>
<td>Locating Single Element Logical Faults along a Path</td>
<td>226</td>
</tr>
<tr>
<td>5.9</td>
<td>Single Control Fault Tolerance of an HT</td>
<td>231</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

In designing a highly concurrent, parallel, or distributed information processing system, a typical problem is the following. Suppose that every source module requests a message from one of the destination modules in the system, how are the messages sent from source modules to destination modules? The answer to this question is not difficult. To satisfy all destination modules, we need a direct or indirect communication path from every source to every destination. But, how should the communication paths be provided in a cost-effective, reliable way to efficiently and flexibly satisfy all types of connection and communication requests to increase the speed and maximize the capability, throughput, and utilization of the system? This dissertation covers some of the theoretical considerations related to this problem.

1.1. General Background

Along the history of computer evolution, computer systems have always reached a level of cost-effective performance or reliability beyond that determined by the contemporary hardware technology
through architectural means. Several evolutionary principles can be identified as pipelining, modular organization, memory interleaving, storage hierarchy, associative processing, reconfiguring, data flow organization, computer networking, etc., among others, from many innovative architectures. A common characteristic of these principles is that they all adopt a certain degree of concurrency, that is, the capability to perform operations which overlap in time. As a result, in current systems either proposed or built by synergizing these architecture principles, different functions are distributed among various resource modules and the same functions are realized by an ensemble of identical resource modules operating in parallel. These systems are commonly referred to as parallel or distributed architectures.

The demand of extremely high performance computer in both real-time and nonreal-time applications coupled with the advance of LSI and VLSI technology has made parallel and distributed architectures containing a large number of modules very popular today [Bern82], [Kart82]. It is conceivable that effective interconnection networks capable of providing communication paths among the large number of available resource modules are very essential in the overall design decisions of parallel or distributed architectures.
Parallelism or functional distribution can be exploited at various levels of computer architecture. Literally hundreds of parallel or distributed structures have been proposed, and many have been built. For surveys, see [Ande75], [Feng77], [Lipo78], [Kuck78], [Kart78], [Mead80], [Kung80], [Kuhn81], [Fern81], [Siew82], [Hayn82]. Some typical architectures having great impact on interconnection networks are given as follows.

An array processor is an SIMD machine [Flyn72] which consists of $n$ identical processing units executing the same instruction broadcasted from a centralized control unit simultaneously. In ILLIAC IV [Barn68], which had a tremendous impact on parallel processing, there are 64 units interconnected in a grid fashion as shown in Figure 1.1(a). Figure 1.1(b) shows an equivalent diagram. The mesh-connected structure allows a processing unit to access data in its four neighboring units directly. However, many applications require data to be routed from processing units which are topologically far apart [Prep81]. The interprocessor communication is considered to be a bottleneck affecting the overall system performance [Agra82a].

As another example, Figure 1.2 outlines the generalized block diagram of Burroughs Scientific Processor (BSP) [Lawr82]. Data come from the memory units in the form of vectors of $n$ or fewer words.
Figure 1.1. Interconnection Scheme of ILLIC IV
The first network does the necessary rearranging or alignment of the elements of the vector so that each element can be paired in some ALU's with the corresponding element of the second vector. The resulting vector of ALU's may be aligned for storage by the second network and stored in the memory units. The alignment networks in BSP are in reality crossbar switches which can realize any arbitrary permutation of data. However, when the number of ALU's becomes large, other networks [Lawr75], [Swan74] have been investigated for cost-effectiveness.

An associative processor is exemplified by STARAN [Bate74] in which each processing element performs bit-level operations. The processing elements communicate through a network called the flip network which permits maximally 256 bits to be permuted as a whole. The flip network permutations are particularly useful for Fast Fourier Transformations and a number of other applications. Figure 1.3 shows a general scheme where the network is suitable for data manipulation [Feng74] in an associative processor.

A multiprocessor [Jone80] is an MIMD machine [Flyn72] which includes autonomous processors with or without local memory and all processors share most, and often all, of the primary memory. Interconnection networks which provide both processor-memory communication and processor-processor synchronization play important
Figure 1.2. Outline of Burroughs Scientific Processor
Figure 1.3. General Scheme of Associative Processors
role. Figure 1.4 gives two general schemes.

The importance of interconnection network can also be found in data flow architectures [Hayn82], special-purpose processors [Kung80], adaptable architectures [Vick80], and multicomputer architectures [Ande75], [Wu82].

1.2. Definitions

In this section, we abstract what is commonly referred as an interconnection network. In particular, the binary element multistage interconnection network and the permutation network, which will be the main concern of this dissertation, are defined as two representative classes of interconnection networks.

1.2.1. Interconnection Networks

An interconnection network (ICN) provides physical connection and communication between the terminals in a specified set of terminals. In Figure 1.5, there are $n$ terminals, $T_1, T_2, \ldots, T_n$. The connection and communication status of the set of the terminals of an ICN at a particular time is its state (function) at that time. The connection and communication status must be physically allowed (realizable), and there should be no two terminals sending messages to each other at the same time (crosstalk-free or interference-free). The connection status can be specified by a
Figure 1.4. Interconnection Schemes for Multiprocessors
partition of the set of terminals into $k$ disjoint subsets, $A_1, A_2, \ldots, A_k$, no $A_i$ empty. A subset $A_i$ with $|A_i| = 1$ represents the set of one disabled (idle or inactive) terminal, and a subset $A_i$ with $|A_i| > 1$ assumes a connection for the terminals in the subset. The communication status can then be specified for each $A_i$ with $|A_i| > 1$ by assigning one terminal in $A_i$ to be the input terminal (sender) and the remaining terminals to be output terminals (receivers). For example, we define a partitioning network as an ICN which has states including all meaningful partitions of the set of terminals and allows all types of communication status. Of course, there may or may not be a state in an ICN realizing an arbitrary partition of the set of terminals of that ICN. For instance, a common bus can be treated as an ICN, and the only one partition possible is the set of terminals itself without subdivision and the communication only allows one terminal to act as sender at one time. Depending on the implementation, there may be more than one method inside an ICN to achieve a state of the ICN (Our definition of state is from a "top-down" point of view). Depending on the state of an ICN, a device connected to a terminal can act as a sender or a receiver for logical message communication purposes, or it can be disabled from the network. For example, an ICN can be used in various high-speed

$1 |X|$ denotes the cardinality of the set $X$. 

computer systems, and the modules (devices) connected to the terminals can be identified as functional units such as processing elements (PE) or memory modules (MM) in the outlining of the general architecture.

Three basic types of functions of an ICN, which correspond to three types of message communication (data manipulation) are implied in the definition of the state. The existence of idle terminals may be achieved by a masking function which essentially prevents the idle terminals from communicating or interfering with the active terminals. A subset $A_i$ with $|A_i|>2$ expresses the multicasting (replicating) function which means a terminal sends a message to more than one terminal at the same time. A common bus essentially realizes broadcasting function which is a special case of the multicasting function. If all $A_i$'s have $|A_i|=2$, a permuting (one-one communicating) or permutation function is assumed from the subset of senders to the subset of receivers.

An interconnection network is two-sided if its terminals are divided into two disjoint subsets, L and R, and no connection through the network between any pair of terminals inside each subset is possible. A terminal is called an inlet (input terminal, or source) if it is not allowed to receive messages from the network. In contrast, a terminal is called an outlet (output terminal, or
Figure 1.5. Interconnection Network

Figure 1.6. One-Way Two-Sided Interconnection Network

Figure 1.7. Two States of a Binary Element
destination) if it is not allowed to send messages into the network.
A bidirectional terminal is one being neither an inlet nor an outlet. A two-sided ICN is said to be one-way if one of the two disjoint subsets of the terminals, say L, contains only inlets and the other subset, R, contains only outlets, and communication is allowed only from the inlets to the outlets through the ICN. Figure 1.6 shows a one-way two-sided ICN with inlets $I_1, I_2, \ldots, I_n$ and outlets $O_1, O_2, \ldots, O_n$. A two-sided interconnection network is said to be pairwise-connected (one-one-connected) if at most one output terminal can be assigned to any given input terminal at one time. A pairwise-connected one-way two-sided ICN can provide the facility of simultaneous communication between $n$ pairs of input and output terminals, $n$ being the minimum of the number of inlets and the number of outlets.

1.2.2. Binary Element Multistage Interconnection Networks

The topology (connection structure) of interconnection networks may be static or dynamic [Feng81]. A static topology has only one connection status, assuming no idle terminals, which is common to all its states, and it may consist only of dedicated links between terminals or may be a multiple bus structure. In this dissertation, we deal mainly with interconnection networks of dynamic topology, dynamic interconnection networks, which contain switches that can
reconfigure communication paths and are capable of providing more than one connection status. A dynamic ICN can be decomposed into (or synthesized by) subnetworks consisting of subsets of terminals interconnected by some created intermediate terminals. When an interconnection network is decomposed into or synthesized by regular columns (stages) of identical switching elements (primitive subnetworks) and a fixed (static) interconnection (linking) pattern is provided for terminals and/or intermediate terminals between adjacent stages or within a stage, it is called a multistage interconnection network (MIN). Obviously, the controlling of the states of a MIN is achieved by setting the states of all of its subnetworks. There may be a number of ways to achieve a state of a MIN. It is possible that a two-way MIN has no state to realize a random assignment of the outlets to inlets.

The two inlet by two outlet switching element capable of straight (state 0) and crossed (state 1) connections is called the binary element, or beta element [Joel68] (see Figure 1.7). An \( n \times m \) binary element multistage interconnection network (\( n \times m \) BEMIN or \( n \) by \( n \), \( m \) stage BEMIN), Figure 1.8, is defined as a pairwise-connected one-way two-sided MIN which has \( n \) inlets and \( n \) outlets (\( n \) is even) and \( m \) stages of switching elements, \( S_1, S_2, \ldots, S_m \), each using beta elements exclusively, and \( m+1 \) link patterns, \( F_0, F_1, \ldots, F_m \). As an example, Figure 1.9 is an 8×5 BEMIN which is known as the 8 by 8
Benes binary network. The specification of a state without idle terminals of an \( n^2 \) \( \text{BEMIN} \) \( \{ \{I_{j_1,0_1}\}, \{I_{j_2,0_2}\}, \ldots, \{I_{j_n,0_n}\}\} \), where \( (j_1, \ldots, j_n) \) is a permutation (bijection) of \( (1, \ldots, n) \), can be simplified to be \( (j_1, \ldots, j_n) \). With inlets labeled 1, 2, \ldots, \( n \) from top to bottom, a state \( (j_1, \ldots, j_n) \) realizes a permutation \( (j_1, \ldots, j_n) \) of inlets \( (1, \ldots, n) \) at the outlets. For our discussion, we limit ourselves to the \( n^2 \) \( \text{BEMIN} \) where each stage contains exactly \( n/2 \) binary elements and each link pattern is a bijection link pattern.

1.2.3. Permutation Networks

A permutation (rearrangeable) network [Bene65] is an \( n \) by \( n \) two-sided ICN whose permitted states realize every one-to-one assignment of outlets to inlets. In other words, suppose both the inlets and the outlets are labeled in an arbitrary way from 1 to \( n \), then it can connect all pairs of input terminals and output terminals with the same label simultaneously. It is then possible in asynchronous mode to rearrange existing connections so as to put in any new desired connection between a pair of idle inlets and outlets. As an example, it has been shown that the Benes binary networks are rearrangeable. In either synchronous or asynchronous operation, high parallelism is available because all types of permuting communication are provided between two sets of modules.
Figure 1.8. The $n^m$ BEMIN
Figure 1.9. Benes Binary Network of Size 8
connected to the inlets and the outlets, respectively, of the network.

In this dissertation, the main results derived are pertinent to using BEMINs as a means to perform permutation network functions.

1.3. A Brief Survey

Many survey papers have been published for interconnection networks [Thur74], [Marc77], [Joel79], [Mass79], [Sieg79], [Feng81]. Our brief survey here gives some background information as well as serves as a motivation for the study of this dissertation. In later chapters, related works and the works on which our study is based are given.

Dynamic interconnection networks are called connecting networks or switching networks which have been studied for long in telecommunication fields [Marc77]. For a connecting network with n inlets and n outlets, it is shown [Clos53] that in order to connect any idle pair of inlets and outlets without affecting any existing connections, it is not necessary to use an n by n crossbar switch. A three-stage Clos nonblocking network is demonstrated to have asymptotically less crosspoints (or binary elements) than those of a crossbar switch. In [Bene65], the fact justifies an idea that even fewer binary elements are needed if rearrangements of existing
connections are allowed, based on the Slepian-Duguid theorem [Dugu59], [Paul62]. Between strict sense nonblocking and rearrangeable networks, a network is nonblocking in the wide sense if suitably choosing routes for new calls it is possible to avoid blocking conditions and still satisfy all demands for connection as they arrive, without rearranging existing calls. In the structure of three-stage Clos networks, the size of middle-stage networks determines a network to be strict-sense nonblocking, wide-sense nonblocking, rearrangeable, or blocking.

Computer architects have proposed for using the BEMINs under Benes rearrangeable network structure, [Lenf78] for example. Are there other structures of link patterns which are not under the Benes rearrangeable structure yet still constitute permutation networks and have better properties? Many blocking networks using BEMINs have been proposed and some are actually built into a parallel computer. Among these are the modified data manipulator [Feng79], the indirect binary n-cube network [Peas77], the banyan (S=F=2) network [Goke73], the reverse-exchange network [Feng79], the reverse baseline network [Feng79], and the omega network [Lawr75], the delta network [Pate79], the flip network [Batc76], and the zeta network [Kim81]. Some general relations for these networks has been found [Feng79], [Krus82]. As a matter of fact, all these networks are under the structure of partial transversal networks (see Chapter
Other BEMINs are proposed for better reliability or fault-tolerance properties [Dias82], [Adam82] or better capability (the number of realizable permutations) with easy control (the setting of the states of binary elements) [Yew81] (see Chapter 5 also).

Given these large number of BEMINs proposed in the literature, little is known in general on the relation of the link patterns of a BEMIN and the set of permutations it can realize. How do the link patterns separate the permutations realizable from those not? In Chapter 2, we study the complexity of the problem whether a given BEMIN with all its link patterns specified can realize a given permutation. This issue is further pursued in Chapter 3 and two major classes of binary element multistage interconnection networks are developed.

A BEMIN which cannot realize all permutations in one pass usually is required to realize an arbitrary permutation in a number of passes. A great amount of work has been concentrated on single-stage BEMIN with perfect shuffle pattern (see Chapter 4). In general, the effect of link patterns on realizing permutations by this single-stage or multistage circulating network is not well understood. Chapter 4 studies problems related to this topic. The
Impact of link patterns of some classes of BEMINs on fault diagnosis and fault tolerance is reviewed and studied for new results in Chapter 5.

A two-sided network with $m_1 \neq m_2$, where $m_1$ and $m_2$ are the number of terminals in the subset L and R respectively, is called a concentration network [Marc77]. A BEMIN with size $n$, $n = \max\{m_1, m_2\}$, can be used as a concentration network. A BEMIN with permutation functions can also be extended to have larger capacity. A full switch [Gecs77] is defined as a partitioning network where the subsets in the partition for the connection status specification are always of size 2; that is, it is capable of performing all meaningful one-one communications for all terminals. If two-way communication is allowed (e.g. by using bidirectional terminals), a full switch as shown in Figure 1.10, and a partitioning network, Figure 1.11, can be constructed with asymptotically minimum number of binary elements [Chun79]. A generalized connection network (GCN) is a one-way two-sided ICN with $n$ inlets and $n$ outlets that can be set to pass any of the $n^n$ mappings of outlets onto inlets. A GCN is constructed in [Thom78] by cascading a generalizer with a connector, Figure 1.12. The generalizer produces the correct number of copies of each of the inputs and can be implemented by a BEMIN with two additional states for each binary element as shown in Figure 1.13. The connector can simply be a BEMIN which is a permutation network.
and it permutes the outputs of the generalizer to the outlets of GCN. If the binary element is extended to have comparison capability for its two input data to be sent separately to the two outlets, Figure 1.14, a BEMIN can realize the functions of a sorting network [Batc68], [Kmut73].

Researches have been conducted to study related topics of BEMINs in general. Graph theory [Catt79] and group theory [Bene75] have been applied to rearrangeability study of connecting networks. A switching circuit theory framework [Prad80] has been used to classify multistage BEMINs. In [Abid80], bit matrices have been used to represent the link patterns of a class of BEMINs, and a simple algebra based on matrices with Kronecker product structure has been used to describe a class of shuffle-exchange networks [Davi81]. The interesting structure of BEMINs has recently been used to study for the VLSI circuit layout complexity [Fran80], [Stei81], for the complexity of parallel computation models [Gali83], and for the time-space trade-offs of problem complexity [Leng82].
Figure 1.10. A Full Switch

Figure 1.11. A Partitioning Network
Figure 1.12. A Generalized Connection Network

Figure 1.13. Two Extended States for a Binary Element

Figure 1.14. Binary Element with Comparison Capability
1.4. The Objective of Investigation

As stated in the opening statements, there are many design objective parameters one must confront. Furthermore, these design parameters are not mutually independent. Any one who tries to find an optimal solution may find himself ending up with a trial and error process in many situations. In some situations, a systematic way may help the designer to make decisions more effectively. However, in some cases, finding an efficient systematic procedure tends to be intractable when the size of the problem becomes large. These situations should be identified with some theoretical supports. We review some basics of what we mean by "intractability" here (referring to [Gare79] for details).

A central question which computer scientists usually ask is the question of determining what can be automated (computed by a computer program). There are two levels to this question, computability and computational complexity. Our interest is the latter which addresses the issue of what can be computed in a practical or reasonable amount of time.

A problem is a question to be answered, a requirement to be fulfilled, or a best possible situation or structure to be found, called a solution, usually possessing several input parameters or variables, which are described but whose values are left
A decision problem is one which requires a simple "yes" or "no" answer, and is no harder (usually also no easier) than its corresponding optimization problem. An instance of a problem is obtained by specifying particular values for all the problem parameters. An algorithm is a step-by-step procedure which when applied to any instance of a problem provides a solution.

The (time) complexity for an algorithm expresses its time requirement by giving, for each possible input length (the size of the problem), the largest amount of time needed by the algorithm to solve a problem instance of that size. A polynomial (exponential) time algorithm is one whose complexity is (is not bounded by, respectively) some polynomial function of the input length. This "polynomial-time" is independent of models of computation and reasonable problem representations for problem instances. Note also that the most efficient parallel algorithm which solves a problem which has no polynomial-time solution, in parallel polynomial-time requires at least exponential number of processors. A problem is intractable if it is so hard that no polynomial time algorithm can possibly solve it.

The status of an algorithm consists of the current values of all variables and the location of the current instruction to be executed. A deterministic algorithm is one for which each status
upon execution of the instruction uniquely determines at most one next status. A **nondeterministic algorithm** is one for which a status may determine many next status and which follows up on each of the next status simultaneously. A problem is in the class P (NP) if there exists a deterministic (nondeterministic, respectively) polynomial time algorithm which solves it.

It is an open question whether P=NP. The class of NP-complete (NP-hard) problems consists of problems in NP (not necessarily in NP, respectively) and not known in P, and all equivalent in the sense that a polynomial time algorithm for any of the problems in the class implies that P=NP. Since there are many NP-complete problems and they are quite different from one another, most researchers conjecture that the NP-complete problems are all intractable, and the knowledge that a problem is NP-complete suggests, at the very least, that a major breakthrough will be needed to solve it with a polynomial time algorithm.

A designer tries to find a network with good property. The design must be aimed at achieving low cost, large capability, high efficiency, easy fault diagnosis, and good fault tolerance. The cost is measured by the number of binary elements or the number of stages of the BEMIN. The capability is the type and the number of permutations the BEMIN can realize. Generally speaking, a network
with large capability has high bandwidth and low blocking probability. The efficiency concerns two parts, the control and the delay. The delay is roughly defined to be the minimum number of binary elements separating any input-output pair under an prespecified, e.g. worst or average, condition. The control is the time of setting binary elements in order to realize a particular permutation or the maximum time to connect an idle inlet-outlet pair. Fault diagnosis is the method to determine whether the network is faulty and to locate the faulty components. Fault tolerance is the capacity of the network to functionally meet a prespecified capability and efficiency under the circumstance which faulty components occur in the network.

As pointed out previously, the number of modules in a system to be interconnected tends to be very large. It is natural to analyze the complexity of some design parameters while assuming other parameters are fixed. In this dissertation, we study dynamic interconnection networks with permutation functions in general by firstly proposing some problems which reflect the complexity of some design parameters. If a proposed problem is found to be solvable in polynomial time, we will try to find a most efficient algorithm to solve it and if it is not likely to be tractable, we will try to show that it is NP-complete or NP-hard. In the mean time, we will also try to derive good properties for some classes of networks
under consideration.

In comparison with the ad hoc analysis techniques and simulations previously used in the literature, our algorithmic approach to study dynamic interconnection networks in general is a new attempt. Nevertheless, the algorithmic approach is not independent of other studies. For example, graph theory and group theory are used in many cases of later chapters to either explore some possibilities or prove a fact. Because either finding an efficient algorithm or proving the NP-completeness requires some deep insight into the structure of the problems and hence the networks under consideration, through the process of algorithmic analysis, old results are seen to be special cases of some generalized uniform results and many new results are obtained. These results will be good references to both theoretical researchers and practical designers.

The problems discussed in this dissertation are organized into four chapters, Chapter 2, 3, 4, and 5, and the summary section in each chapter outlines the content of that chapter. Chapter 6 concludes the current research and presents further research problems and directions.
An interconnection network can be decomposed into a number of switching elements connected by intermediate terminals (links). To synthesize a general network with a single type of switching element, the simplest and the most interesting single element is the 2 by 2 element. The 1 by 1 2-state element is the most primitive element, but it must be combined with fan-in and fan-out elements (elements with unequal number of inlets and outlets). Many complex and interesting networks can be constructed by using exclusively 2 by 2 elements [Hopp79], [Jans80], [Feng81]. In this chapter we limit ourselves to discuss the 2-state binary elements, and we will focus on the link patterns between two columns of binary elements in a BEMIN.

2.1. The Particular Permutation Realization (PPR) Problem

Partly because the number of possible link patterns is exponential, the relation between the link patterns chosen to build a network and the permutations it can realize is not clear. Whether a given interconnection pattern is more versatile in realizing
permuting functions than another is unknown to us. This chapter studies interconnection link patterns in general from an algorithmic viewpoint. More specifically, we consider the existence of an efficient algorithm to test the permutation realization properties of a network given its interconnection link patterns.

The PPR problem: Given an \( n \times m \) BEMIN, determine if it can realize a given permutation of the inlets at the outlets.

We will try to decide whether the problem is polynomially decidable or it is an NP-complete problem [Gare79]. We say that a problem is easy when it can be solved in polynomial time; we say hard when we mean NP-complete or NP-hard. For convenience, in this chapter we assume that the link patterns \( F_0 \) and \( F_m \) in the \( n \times m \) BEMIN are the identity permutation and no \( F_k, 1 \leq k \leq m-1 \) is the identity.

**Lemma 2.1:** PPR is in NP.

**Proof:** A simple nondeterministic algorithm just guesses the state of each beta element and checks if the network can achieve the desired permutation. This is shown by the following algorithm.

**Algorithm A:**

Input. An \( m \) stage \( n \) by \( n \) BEMIN with link patterns \( F_1, F_2, \ldots, F_m \), where \( F_i=(f_1, \ldots, f_n) \) is a permutation of \( (1,2,\ldots,n) \) (optionally, let \( f_i > f_{i+1} \) for all \( i \) odd). A given permutation \( P=(p_1, \ldots, p_n) \) of
Output. "Success" if the BEMIN can realize P. "Failure" if the BEMIN cannot realize P.

Method.

Procedure PR (m,n,F_l,...,F_m,P);

var
  BEMIN[m,n/2]:Boolean Array;
  LABEL[n]:Integer Array;
begin
  for i:=1 to m do
    for j:=1 to n/2 do
      BEMIN[i,j]:=choice (true, false);
      /*guess the state of each binary element*/
  for i:=1 to n do LABEL[i]:=i ;
  for i:=1 to m do
    begin
      permute according to pattern F_i ;
      if BEMIN[i,j] then exchange LABEL[j*2] and LABEL[j*2-1] ;
      end ;
    if LABEL=P then "success" else "failure" ;
    /*check the realizability*/
  end PR ;

Q.E.D.

One can generate all the possible states of the n*m BEMIN by simulating the nondeterministic Algorithm A, and check the permutation P against the list of states, but clearly it will take $O(2mn)$ time. There are two natural directions to pursue that the PPR problem is either polynomially solvable or NP-complete. One is the time needed relative to the increment of the number of stages m. The other is the time needed relative to the increment of the number of input and output terminals n. One can try to see if adding one stage (fixing n) or adding one row of elements (fixing m) would
"double" (or "multiply") the work of algorithmic analysis.

2.2. Some Polynomial Solutions

In this section we will prove that several special cases of the PPR problem are solvable in polynomial time.

Theorem 2.2: It takes only polynomial time to solve PPR in case n is fixed.

Proof: There exists a polynomial time divide-and-conquer algorithm as follows which takes advantage of the fact that n is fixed.

Algorithm B:

Let I=(1,2,...,n).
If DAC(m-stage BEMIN, I --> P) then output "yes" else output "no".

Define PROCEDURE DAC(m-stage BEMIN, I --> P);
1. If m=1 then return "true" or "false" in constant time.
2. Divide the network into left half (LH) and right half (RH) by cutting in the middle.
3. Test for all n-permutations of (1,2,...,n).
   (1) Get next permutation K=(k1,k2,...,kn).
   (2) if DAC(LH, I --> K) then continue, or else goto (1).
   (3) if DAC(RH, K --> P) then return ("true"), or else goto (1).
4. return ("false").
END DAC.

Since the solution to $T(n) = k T(n/c) + f(n)$ is $T(n) = n \log_{ck} [T(1) + \sum_{i=1}^{\log_{ck}n} f(ci)]$ [Bent78],¹ we have

¹SIGMA denotes the summation notation.
the time for the algorithm

\[ T(m) = O(m \log(2^n)) \]

Q.E.D.

Now we move to the other direction where we will fix the stage variable \( m \).

**Theorem 2.3:** For an \( n \times m \) BEMIN with fixed \( m \), it is easy to decide whether \( k \) (fixed) inlets can be simultaneously connected to \( k \) desired outlets.

**Proof:** It is obvious that the problem can be solved in polynomial time since both \( m \) and \( k \) do not change when \( n \) increases. An algorithm can simply test for all possible paths from the \( k \) inlets to the \( k \) outlets. Q.E.D.

We note that the result in [Fort80] for directed acyclic graphs implies that there exists an efficient algorithm which solves in \( O(m^2n) \) time and \( O(n^2k) \) time for the problem in Theorem 2.2 and Theorem 2.3 respectively.

If a BEMIN has the answer "yes" in the problem of Theorem 2.3, we say that the network has **bunch connectivity of size** \( k \) for the \( k \)

\(^1\log\) denotes logarithm base 2.
inlets and k outlets. Rearrangeability of an n*m BEMIN then means bunch connectivity of size n for all possible n! permutations.

A two-sided IN is said to have **terminal-to-terminal connectivity** for a permutation \( P = (p_1, \ldots, p_n) \) iff there is a state of the network such that a path exists to connect any input terminal \( i \) to the desired output terminal \( p_i \). Apparently, we can test the terminal-to-terminal connectivity first in polynomial time for the PPR problem. This problem can actually be a simple path-finding problem such as the shortest path problem. For 2-stage BEMIN, it turns out that this is all we need.

**Theorem 2.4**: The PPR problem for n*m BEMIN is easy in case m=2 or 3.

**Proof**: (1) 2-stage case.
There is at most one path from an inlet to an outlet except the situation of Figure 2.1. In Figure 2.1, it does not matter which path one choose for the desired connection. Thus, one can simply test the terminal-to-terminal connectivity of the network for the permutation.

(2) 3-stage case.

1iff denotes if and only if.
Figure 2.1. The 2-Stage Case

Figure 2.2. The 3-Stage Case
If it is possible for any two inlets(outlets) connected to the same inlet(outlet) switch (the first(last) stage switch) to be routed simultaneously to the desired outlets(inlets, respectively) according to a permutation P, then we say that the network has element-to-element connectivity for the permutation P. It turns out that we only need to check for element-to-element connectivity to decide this case.

We will begin by finding those first stage and last stage beta elements with single way of choosing links to be connected to the desired outlets or inlets. If the connectivity fails, then the algorithm ends and gives the answer to the problem "no". Secondly, if we delete these connections, the remaining network can only have the structure shown in Figure 2.2. One can easily see that this structure is rearrangeable. Since we only delete those connections which can be accomplished in only one way, the deletion step does not lose the generality of simply testing element-to-element connectivity. From Theorem 2.3, the proof is thus completed. Q.E.D.

The technique used in the proof of Theorem 2.4 does not work for the 4-stage case. A counterexample is shown in Figure 2.3. The network has both terminal-to-terminal and element-to-element connectivity, but the permutation P is not realizable.
Figure 2.3. A Counterexample
2.3. Problems Related to PPR

The fact that finding an efficient algorithm for the PPR problem is not easy makes us to suspect that the problem could be NP-complete. In this section, we will find some related problems in graph theory and other related problems close to what we need to show the NP-completeness of PPR. The relation we would use is "reduction". The fact that a problem Q is polynomially reducible to a problem Q' implies that if Q' is easy then Q is easy and if Q is hard then Q' is hard. If two problems can be reduced to each other, then they are "polynomially equivalent". See [Gare79].

2.3.1. Problems Equivalent to PPR

A digraph G is defined to be a pair \((V,E)\) where \(V\) is a set \(\{v_1, v_2, \ldots, v_n\}\) of elements called vertices(nodes), and \(E\) is a family (or system) \((e_1, e_2, \ldots, e_m)\) of elements of the Cartesian product \(V \times V\), called edges(arcs). A vertex \(v_i\) can be drawn as a point denoted \(v_i\) and an edge \(e_j=(v_i, v_i')\) as an arrow denoted \(e_j\) joining the point \(v_i\) to \(v_i'\). It is convenient, whenever it is not ambiguous, to remove the directions of the arrows of a digraph G. In fact, this underlying graph of G is actually an (undirected) multigraph.

A digraph \(G=(V,E)\) constructed from an \(n \times m\) BEMIN by making each inlet, each outlet, and each link, an edge heading to the output
side from the input side and making each binary element a vertex, together with the vertices created by the edges corresponding to the inlets and the outlets is called a BEMIN acyclic digraph. Thus, the BEMIN acyclic digraph corresponding to the n*m BEMIN has \((m/2+2)n\) nodes and \((m+1)n\) arcs and it has no directed cycles (circuits). We call the nodes with indegree 0 sources (input nodes), and the nodes with outdegree 0 destinations (output nodes). There are n sources and n destinations.

A digraph \(G=(V,E)\) constructed from a BEMIN and a particular permutation \(P\) by connecting an arc from each output terminal to its desired input terminal according to \(P\), and making each beta element a vertex and each link an arc heading to the output side from the input side, is called a BEMIN cyclic digraph. Thus, a BEMIN graph is a \(4-\)regular (2-in, 2-out) Eulerian digraph which has \(mn/2\) nodes and \(mn\) edges. We call \(m\) the width and \(n/2\) the height of the BEMIN digraphs being either cyclic or acyclic.

If each vertex in a BEMIN acyclic graph is "replaced" by a copy of the small 4-vertex digraph of Figure 2.4, and from the source side to the destination side the vertices for the links of the digraph are "contracted" by the method shown in Figure 2.5, then the resulting graph is called the VBMIN acyclic digraph. It should be pointed out that the contraction can also be done for a replaced
stage of binary elements with its left-hand side link pattern instead of the right-hand side one, and the contraction process does not necessarily follow the direction from the source side to the destination side; it can start with any stage and operate from that stage to the both sides. These replacement and contraction operations will give the **VBMIN cyclic digraph** when applied to a **BEMIN cyclic digraph**. An example is shown in Figure 2.6.

**Theorem 2.5:** The PPR problem is equivalent to the following problems.

(a) With sources labeled $1, 2, \ldots, n$ and destinations labeled according to the permutation $P$, determine if there are $n$ edge-disjoint paths which match the sources to the destinations in the corresponding **BEMIN acyclic digraph**.

(b) Determine if there are $n$ edge-disjoint circuits in the corresponding **BEMIN cyclic digraph**.

(c) With sources labeled $1, 2, \ldots, n$ and destinations labeled according to the permutation $P$, determine if there are $n$ vertex-disjoint paths which match the sources to the destinations in the corresponding **VBMIN acyclic digraph**.

(d) Determine if there are $n$ node-disjoint cycles in the corresponding **VBMIN cyclic digraph**.

(e) The PPR problem restricted to the case in which the permutation is the identity.
Figure 2.4. The 4-Vertex Digraph

Figure 2.5. An Example of Contraction
A BEMIN cyclic digraph

The corresponding VBMIN cyclic digraph

Figure 2.6. An Example of BEMIN and VBMIN Cyclic Digraphs
Proof: Problems (a), (b), (c), and (d) are obvious from the above definitions. Problem (e) is proved by reduction from PPR to the restricted identity PPR as follows.

For every instance of PPR, we construct an instance of the identity PPR by adding one stage to the BEMIN at the output side so that the link pattern $F_m$ follows that of $P^{-1}$. If the $m$-stage BEMIN of PPR is realizable for $P$, then it is obvious that the identity PPR consisting of the $(m+1)$-stage BEMIN is realizable by setting all elements of the last stage at state 0. Conversely, if the $(m+1)$-stage BEMIN of the identity PPR has a state to realize the identity permutation, then there is a state setting for every element of the $n*(m+1)$ BEMIN. If every element in the last stage is set to state 0, then the $m$-stage BEMIN of PPR will realize $P$ by the state setting of the first $m$ stages of the $(m+1)$-stage BEMIN. For any element set to state 1 in the last stage, we know that there are two edge-disjoint paths connecting the corresponding first stage element to this element. By complementing the state of both elements, the identity permutation is still realizable. Thus, every element in the last stage can be reset to state 0 and $P$ is realizable for the $m$-stage BEMIN of PPR. Q.E.D.
2.3.2. The Vertex-Version of PPR (VPPR) Problem

We notice that the VBMIN digraph has the property of containing only alternating 4-cycles (cycles of length 4) for the connecting patterns. Figure 2.7 classifies the various linking types by the alternating 2*k-cycles, k=1,2,..., for the connecting patterns between stages. Each 2*k-cycle contains a class of topologically equivalent link patterns (see Chapter 3). We note that even though each link pattern is topologically equivalent for two given 2-sided MINs, the two networks may not be topologically equivalent. Now we will consider the more generalized cases of the vertex-version problem, that is, we not only allow 2*2-cycles in the graph, but 2*1-cycles, 2*3-cycles, 2*4-cycles, ..., and 2*n-cycles are allowed. The graphs associated with these problems are called VAMIN acyclic digraph and VAMIN cyclic digraph.

There is a corresponding type of interconnection network (VAMIN) to the VAMIN digraph. Figure 2.8(b) shows the network corresponding to the graph of Figure 2.8(a). The switching element can be viewed as that of Figure 2.9, which can be implemented by a 2-by-1 multiplexor cascaded by a 1-by-2 demultiplexor [Haye78]. The switching element only allows one signal to pass at a time and it has 4 states. VAMINs may be more flexible than BEMINs, but they may cost more. Another way to interpret the VAMIN graph is to treat
Figure 2.7. Elementary Linking Types
edges as crosspoint switches and nodes as terminals [Cant72]. The corresponding problem is stated as follows.

The VPPR problem: Given an n×m VAMIN and a permutation $P = (p_1, \ldots, p_n)$, determine if the VAMIN can realize $P$.

It is clear that PPR is a special case of VPPR. Also, we have the following similar result.

Theorem 2.6: The VPPR problem is equivalent to the following problems.

(a) With sources labeled 1, 2, ..., n and destinations labeled according to the permutation $P$, determine if there are $n$ node-disjoint paths which match the sources to the destinations in the corresponding VAMIN acyclic digraph.

(b) Determine if there are $n$ node-disjoint circuits in the corresponding VAMIN cyclic digraph.

(c) The VPPR problem restricted to the case in which the permutation is the identity.

Proof: Problems (a) and (b) are obvious from the definitions. Problem (c) is proved by reduction from VPPR to the restricted identity VPPR. For every instance of VPPR, we construct an instance of the identity VPPR by adding one stage to the VAMIN at the output side. The link pattern is defined by first connecting the elements
Figure 2.8. A VAMIN and its VAMIN Acyclic Digraph

Figure 2.9. A Switching Element in VAMIN
at stage $m$ to the elements at stage $m+1$ according to the permutation $P-1$ and then doubling each link connection. Clearly, the $m$-stage VAMIN in VPPR can realize $P$ if and only if the $m+1$-stage VAMIN in the identity VPPR problem can realize the identity permutation since there is only one possible connection for the last stage. Q.E.D.

2.3.3. More Related Problems

We note here that VPPR and PPR are restricted cases of the generalized path-finding problems. Also, they are special cases of the subgraph homeomorphism problems [John81]. These problems are related to the integer commodity flow problems [Even76], [Itai78], [Karp75].

The subgraph homeomorphism problem (SHP) is to determine if a graph $G$ contains a subgraph homeomorphic to another graph $H$. That is, a subgraph $G'$ can be converted to $H$ by repeatedly removing any vertex of degree(valence) 2 and adding an edge joining its two neighbors. When $G$ and $H$ are directed graphs then the vertex of degree 2 must have one edge directed in from a vertex $v_i$ and one edge directed out to a vertex $v_j$, and the edge must join the two neighbors in the proper direction from $v_i$ to $v_j$. In other words, the homeomorphism maps nodes of $H$ to nodes of $G$ and edges of $H$ to simple paths, all pairwise node-disjoint, in $G$. There are several versions of SHP. $G$ and $H$ can be either both directed or both
undirected. The graph $H$ can be given as input or it can be fixed as a pattern graph. The node-mapping from $H$ to $G$ may be specified (the fixed-vertex version) or left arbitrary. We will limit ourselves to the case which $G$ and $H$ are both inputs and directed graphs, and the node-mapping is specified. We will call this the RSH problem.

Obviously, the VPPR is equivalent to RSH where $H$ is $n$ disjoint links (a link is an edge which is not a loop), $G$ is the $n \times m$ VAMIN acyclic digraph and the node mapping is specified according to the permutation $P$, or equivalent to RSH where $H$ is $n$ loops and $G$ is the $n \times m$ VAMIN cyclic digraph.

**Theorem 2.7:** When the input graph $H$ consists of $n$ loops, RSH is NP-complete even if $G$ is Eulerian.

**Proof:** We will reduce 3SAT (the satisfiability problem for Boolean Formula in 3-CNF [Aho74]) to this RSH with input graphs $G$ and $H$. We give a construction which, in polynomial time, transforms any instance of 3SAT to an instance of this RSH.

Let the clauses $C_1, \ldots, C_k$ and the literals $x_1, \ldots, x_n, x'_1, \ldots, x'_n$ be an instance of 3SAT. For each variable $x_i$, we construct a subgraph $G_i$ of $G$ as shown in Figure 2.10. Here $p_i$ is the number of occurrences of $x_i$ in the clauses and $q_i$ is that of $x'_i$. The nodes $x_i$ and $x'_i$ are identified as a single node. Nodes with prime constitute the lower part of $G_i$ and nodes without prime constitute
the upper part of \( G_i \). For each clause \( C_j \), we also construct a subgraph of \( G \) as shown in Figure 2.11. Note also that \( C_j \) and \( C_j^* \) are identified. To complete the definition of \( G \), there is an edge from \( s_j \) to \( v_i^{2r-1} (v_i'^{2r-1}) \) and also an edge from \( v_i^{2r} (v_i'^{2r}) \) to \( t_j \), for the \( r \)-th occurrence of \( x_i^r \) (\( x_i'^{r} \)) found in clause \( C_j \). Thus, \( G \) is Eulerian. The construction for \( H \) is simply \( k+h \) disjoint loops. The node mapping is specified by the nodes \( C_1, \ldots, C_k, x_1, \ldots, x_h \).

(a) Assume that there exist \( k+h \) node-disjoint simple circuits through nodes \( C_1, \ldots, C_k, x_1, \ldots, x_h \), respectively. The circuit for node \( x_i \) must pass through either the upper part or the lower part of \( G_i \). Define \( x_i \) to be "true" if and only if the circuit is routed through the lower part. In this case, the circuits for the nodes \( C_1, \ldots, C_k \) may pass through the upper part of \( G_i \). Suppose the circuit for \( C_j \) passes through the upper part of \( G_i \), then from the construction we know that \( x_i \) is in \( C_j \). Thus, \( x_i \) is "true" and \( C_j \) is satisfied. Similarly, if we suppose that the circuit for \( C_j \) passes through the lower part of \( G_i \), then from the construction we know that \( x_i' \) is in \( C_j \). Since \( x_i \) is "false", we have that \( x_i' \) is "true" and \( C_j \) is satisfied.

(b) If the expression is satisfiable, the simple circuit for node \( x_i \) will be routed through the upper part of \( G_i \) if the variable \( x_i \) is assigned to "false" and the lower part if "true". Since in
Figure 2.10. Subgraph for the Variable $x_1$

Figure 2.11. Subgraph for the Clause $C_j$

Figure 2.12. The 3-In and 3-Out Switching Elements
each clause there is at least one literal \( x_i \) or \( x'_i \) which is assigned the value "true", the simple circuit for node \( C_j \) can pass through the upper or lower part of \( G_i \) depending on whether \( x_i \) or \( x'_i \) is "true" and the vertex-disjoint requirement is met. Q.E.D.

The above theorem is really close to what we need to claim that VPPR is NP-complete. However, since \( G \) must contain some nodes with valence of at least 6, the network implied will have to contain elements as that shown in Figure 2.12(a) and Figure 2.12(b). It is easy to disprove that the 3-in and 3-out nodes in the construction can be further reduced to a number of 2-in and 2-out nodes, since any Eulerian digraph with 2-in and 2-out nodes which replaces a 3-in and 3-out node in the constructed graph \( G \) will introduce at least a new node-disjoint circuit for some instances.

The relation between the problem of finding the maximum number of edge-disjoint (and vertex-disjoint) circuits and the problem of finding the minimum number of edges (and vertices, respectively) meeting all circuits is discussed in section 7.
2.4. The VPPR Problem is NP-Complete

From Theorem 2.6, we will assume for discussion that the permutation $P$ is the identity. If an $n \times m$ VAMIN acyclic digraph can realize the identity permutation, this means there are $n$ node-disjoint paths routed from source 1 to destination 1, source 2 to destination 2, ..., and source $n$ to destination $n$. If one tries to find an efficient algorithm, he could test the connectivity for all $n (s_i, d_i)$ pairs. This is polynomial since one can just check for reachable nodes from the first to the last stage for each pair $(s_i, d_i)$. Furthermore, if for an $s_i$, there is only one way of choosing nodes to route it to the corresponding $d_i$, then the path and the edges associated with it can be deleted. This would simplify the problem in some sense.

Now suppose all pairs have multiple ways of choosing nodes to connect them. It is clear that the number of ways to connect a pair is bounded by $2^{m-1}$ if the number of stages is fixed. But the mutually node-disjoint requirement means some connecting paths may prevent the successful connection of other pairs. The property of rearrangeability gives us a clue that one may be able to reroute certain paths and satisfy all the pairs. The routing process needs backtracking. The remaining question is whether we can gradually simplify the network (e.g. removing one stage at a time by examining
a small number of stages around it) by limited backtracking or we have to exhaust all the possible cases. Our results implies that the latter alternative generally applies.

A characterization of the satisfiability problem is obtained in [Scha78] and it is concluded that the following problem is NP-complete.

The ONE-IN-THREE 3SAT problem: Given a set U of Boolean variables, a collection D of clauses over U such that each clause in D contains 3 literals, decide if there is a true assignment for U such that each clause in D has exactly one true literal.

The following theorem will serve us to show the NP-completeness of VPPR and PPR.

Theorem 2.8: The ONE-IN-THREE 3SAT problem is NP-complete. This is true even when for each variable x in U there are at most 5 clauses in D that contain either the literal x or x'.

Proof: The reduction from 3SAT to ONE-IN-THREE 3SAT can be done as follows. For each clause \( C = (u_1, u_2, u_3) \) of an instance of 3SAT, we create 8 new variables \( y_1, \ldots, y_8 \) and construct a set D of 6 clauses corresponds to C. Let \( D = \{(u_1, y_1, y_2), (u_2, y_2, y_3), (u_3, y_4, y_5), (y_1, y_3, y_6), (y_2, y_4, y_7), (y_5, y_8, y'_8)\} \). It is not difficult to check that C is satisfiable if and only if D is satisfiable.
From [Gare79], we have that 3SAT is NP-complete even if for each variable \( x \) in \( U \) there are at most 5 clauses in the collection that contain either the literal \( x \) or \( x' \). In the construction we do not have repetition of old literals and any new literal occurs less than 5 times. Thus, the same result holds for the ONE-IN-THREE 3SAT problem. Q.E.D.

**Theorem 2.9:** VPPR is NP-complete, and it remains NP-complete even if the number of stages \( m \) is fixed and is not less than 7.

**Proof:** We will reduce ONE-IN-THREE 3SAT to VPPR for the identity permutation. The reduction will imply that VPPR is NP-complete even when \( m=8 \).

For each instance of ONE-IN-THREE 3SAT, we construct a VAMIN acyclic digraph. For each clause, there are six source nodes and six destination nodes corresponding to it. Let the clauses \( C_1,...,C_k \) and the literals \( x_1,...,x_h, x'_1,...,x'_h \) be an instance of ONE-IN-THREE 3SAT. There will be \( 6k \) sources and \( 6k \) destinations.

Let the total number of occurrences of the positive and the negative literals of a variable \( x_i \) be \( r_i \). Then for each variable \( x_i \), we will construct a subgraph \( G_i \) of the VAMIN graph corresponding to it. Each \( G_i \) contains a 2-stage with an alternating \( 2*2r_i \)-cycle cascaded by another assistant stage. The link pattern of the
$2^*2r_i$-cycle is defined as a permutation $F_r=(1,4r-1,2,3,\ldots,4r-2,4r)$, where the subscript $i$ of $r_i$ is dropped. The $2^*2r_i$-cycle link pattern has the property that if there are $2r_i$ paths, then if any path chooses the straight (slant) path, then all other paths will have to choose the straight (slant, respectively) paths. In other words, there are exactly 2 matchings for each $2^*2r_i$-cycle.

The link pattern before the assistant stage is defined as follows. For each odd node of the assistant stage from the top, there is a double edge ($2*l$-cycle) from the node of the previous stage at the same level. The pattern of all even nodes between the two stages forms a $2^*r_i$-cycle, and the links are defined by a permutation which equals $F_{r/2}^{-1}$ if the even nodes are labeled 1, 2, ..., $r/2$ from top to bottom. The assistant stage enables paths of $G_i$ to be routed through consecutive pairs $(v_1,v_2),(v_3,v_4),\ldots,(v_{2r-1},v_{2r})$ of nodes, from top to bottom, to $(v_1,v_2),(v_3,v_4),\ldots,(v_{2r-1},v_{2r})$ or to $(v_2,v_1),(v_4,v_3),\ldots,(v_{2r},v_{2r-1})$ depending on whether the straight or the slant path is chosen.

Figure 2.13 shows the variable subgraphs $G_i$'s with occurrence 1, 2, and 3. The single arrow denotes the straight path, and the double arrow denotes the slant path. All $G_i$'s are layout in parallel from top to bottom as stage 4, 5, and 6 of the VAMIN digraph. From top to bottom, every 2 consecutive nodes in $G_i$ at
stage 4 are considered as a pair corresponding to a literal \( u_i \) contained in the clause \( C_j \).

The sources corresponding to the clauses will then be led from the left-hand side of the \( G_i \)'s, through the \( G_i \)'s by passing along the pairs, and then to the destinations at right-hand side of the \( G_i \)'s. The pattern between stage 3 and stage 4 depends on whether the literal is positive or negative. If the literal is negated, the links of the pair are twisted; otherwise, the links are led straight. All edges are doubled between stage 3 and stage 4.

The six source nodes corresponding to a clause \( C_j = (u_a,u_b,u_c) \) are divided into a true-setting pair \( st_j \) and 2 false-setting pairs \( sf_1j \) and \( sf_2j \). Similarly, the 6 destinations are so divided into \( dt_j \), \( df_1j \), and \( df_2j \). The sources and the destinations are arranged at levels according to the occurrence of the literals \( u_a, u_b, \) and \( u_c \) in the collection of clauses. The true-setting pair will be connected through intermediate stages and \( G_a, G_b, \) and \( G_c \) without twisting of the links. The false-setting pairs will be connected through intermediate stages and \( G_a, G_b, \) or \( G_c \) with exactly one twist of the links, after stage 1 or stage 2, and in such a way that all nodes have 2-in and 2-out except that sources have 2-out, and destinations 2-in.

An example for the first clause \((x_1,x_2,x_3')\) is shown in Figure
Figure 2.13. Subgraphs with Occurrence 1, 2, and 3
We now prove that the instance of the ONE-IN-THREE 3SAT is satisfied if and only if the corresponding instance of the VPPR by the above construction is satisfied.

(a) Assume that the expression of the instance of the ONE-IN-THREE 3SAT is satisfiable. Let $G_i$ be routed by the straight paths if the variable $x_i$ is assigned "true" and by the slant paths if the variable $x_i$ is assigned "false". There are $6k$ paths provided in the middle in this way. Then each clause $C_j = (u_a, u_b, u_c)$ has exactly one literal assigned "true", say $u_a$, and the other 2 literals "false". Let the true-setting pair $s_{tj}$ corresponding to this clause pass through the $G_a$ to $dt_j$. This can be done since if $u_a$ is a negated literal, then $s_{tj}$ will have to use the slant paths to be connected to $dt_j$ to achieve the identity mapping, and slant paths are allowed in $G_a$ since $x_a$ is "false". Similarly, it can be done if $u_a$ is non-negated.

The false-setting pairs $sf_{1j}$ and $sf_{2j}$ corresponding to this clause can then pass through $G_b$ and $G_c$ for reasons similar to those above. Notice that each of these two false-setting pairs have a twist before it is led to the $G_i$'s. Clearly, all 6 paths are node-disjoint. This is true for all clauses, and all $6k$ paths are node-disjoint.
Figure 2.14. An Example for \((x_1, x_2, x'_3)\)
(b) If there are 6k vertex-disjoint paths for the 6k*8 VAMIN mapping according to the identity, then there are 6 vertex-disjoint paths routed through corresponding to each clause \( C_j = (u_a, u_b, u_c) \). Let the variable \( x_i \) be assigned "true" ("false") if the corresponding \( G_i \) is passed by straight (slant, respectively) paths.

Clearly, sources \( st_{1j} \) and \( st_{2j} \) in the pair \( st_j \) must pass through the \( G_i \) in which the corresponding literal \( u_i \) is "true" to the destinations \( dt_{1j} \) and \( dt_{2j} \) in the pair \( dt_j \). Furthermore, \( st_1 \) and \( st_2 \) must pass through the same \( G_i \). If not, they separate and pass through 2 \( G_i \)'s. Then only one \( G_i \) is left for \( sf_{3j} \) and \( sf_{4j} \) in \( sf_{1j} \) and \( sf_{5j} \) and \( sf_{6j} \) in \( sf_{2j} \) to be routed to their destinations. Since there are only 2 paths provided, this is a contradiction.

Assume that \( st_j \) passes through \( G_a \), and assume \( u_a \) is a positive (negative) literal. Then \( st_{ij} \) must be routed by straight (slant) paths in \( G_a \) and \( x_a \) is assigned "true" ("false", respectively). Thus, \( u_a \) is "true".

After \( st_{1j} \) and \( st_{2j} \) have been settled, \( sf_{3j} \), \( sf_{4j} \), \( sf_{5j} \), and \( sf_{6j} \) will have to go through \( G_b \) and \( G_c \) to their destinations. Since there is a twist before stage 3 for each pair, the corresponding literal \( u_b \) and \( u_c \) must be "false" by similar reasons as above. Since the argument is true for every clause, the ONE-IN-THREE 3SAT is satisfied. Q.E.D.
2.5. The NP-Completeness of PPR

If a VAMIN acyclic digraph can realizable a permutation $P$, it is clear that only half of the links are required for a connection to realize $P$. When a BEMIN acyclic digraph is able to realize a permutation $P$, it has to use all the links and all paths must be edge-disjoint. It is actually the same as asking if an $n \times m$ BEMIN cyclic digraph can be completely decomposed into $n$ edge-disjoint cycles. If the answer to this question is "yes", then indeed, $n$ is the maximum. This simplified case, however, is still NP-complete.

Theorem 2.10: PPR is NP-complete, and it remains NP-complete even if the number of stages $m$ is fixed.

Proof: We will reduce ONE-IN-THREE 3SAT in Theorem 2.8 to PPR for the identity permutation.

We notice that an $n \times m$ BEMIN cyclic digraph is Eulerian and each vertex has degree 2-in and 2-out. Indeed we can generalize the digraph of the 2-in and 2-out case to an Eulerian one containing nodes with $d$-in and $d$-out for any integer $d$. The reduction of any $d$-in and $d$-out node with $d>2$ is done by replacing the node by a rearrangeable $d \times g$ BEMIN acyclic digraph corresponding to the network construction by [Opfe71] and $g$ will be the integer part of $\log_2 d$. Some examples are shown in Figure 2.15. Since the $d \times g$ BEMIN
is rearrangeable, there are conflict-free paths for any permutation assignment of "in" edges to "out" edges incident to the node replaced, and the routing of the paths for any assignment can be done in polynomial time [Opfe71].

The digraph can then be laid out appropriately with a possible "stretching" of nodes such that each stage has the same number of nodes. By stretching we mean the operation which replaces a d-in and d-out node by two nodes, the d "in" edges incident to the "left" node and the d "out" edges incident to the "right" node, and adds d edges to join the left node to the right node. For those 1-in and 1-out nodes, we combine two of them in the same stage by the technique shown in Figure 2.16. If there is only one vertex of degree 2 left in each stage, then we will make a copy of the whole network and repeat the the above construction.

An example is shown in Figure 2.17. The equivalence shown above allows us to consider the Eulerian-type case for the remaining transformation.

Let the clauses $C_1, \ldots, C_k$ and the literals $x_1, \ldots, x_h, x'_1, \ldots, x'_h$ be an instance of ONE-IN-THREE 3SAT. Let the number of occurrence of the positive and the negative literals of a variable $x_i$ be $r_i$. Then for each variable $x_i$, construct a subgraph $G_i$ similar to that done in the proof of Theorem 2.9. Figure 2.18 shows
Figure 2.15. Reduction Examples for $d$-in,$d$-out ($d>2$)

Figure 2.16. Reduction from 1-in,1-out to 2-in,2-out
Figure 2.17. Reduction for Single 1-in,1-out Node
the $G_i$'s with occurrence 1, 2, and 3. Note the difference in the middle of $G_i$'s.

All $G_i$'s are layout in parallel in the middle. From top to bottom, two consecutive levels are considered as a ("main") pair. To make a $G_i$ Eulerian, "dummy" nodes will further be added for it.

The sources corresponding to the clauses will be led from the left-hand side of the $G_i$'s, through the $G_i$'s by passing the main pairs, and then to the destinations at the right-hand side of the $G_i$'s. A source and a destination will be identified if they are of the same level. The link pattern preceding $G_i$'s depends on the whether literals are being negated or non-negated. If a literal $u_i$ is negated, the corresponding links in front of the $G_i$ will be twisted; otherwise, they will be straight. The $G_i$ has the property that if any main pair choose the straight (slant) path, then all other pairs will have to choose the straight (slant, respectively) paths.

For each clause $C_j=(u_a,u_b,u_c)$, there are 3 pairs of sources $t_j$, $f^1_j$, and $f^2_j$ corresponding to it. The sources will be arranged at the corresponding levels according to the occurrence in $G_a,G_b,G_c$. The pair $t_j$ will be led through $G_a,G_b,G_c$ without twisting of links. The pair $f^1_j$, and $f^2_j$ will be led through by first twisting the links and then combined with the pair $t_j$ to make the construction
Figure 2.18. Examples of Variable Subgraphs
Eulerian. An example for the first clause \((x_1, x'_2, x'_3)\) is shown in Figure 2.19. In the example, the occurrence of \(u_1, u_2, \) or \(u_3\) is 2. The nodes above each \(G_i\) are dummy nodes.

The proof will be similar to that of Theorem 2.9.

(a) Suppose that the expression is satisfied. Then each clause \(C_j = (u_a, u_b, u_c)\) has exactly one literal assigned "true", say \(u_a\), and the other 2 literals "false". Let the \(t_j\) pair pass through \(G_a\) straight or slant depending on that \(u_a\) is non-negated or negated. Assume that \(u_a\) is non-negated.

The \(t_j\) pair, once passed through, will force all other main pairs in \(G_a\) to pass through straight. Since any other occurrence of the \(x_a\) demands such straight paths in \(G_a\), it may pass through accordingly. For any occurrence of \(x'_a\), the corresponding paths must be routed from the \(f_1\) or \(f_2\) pair which can be done since \(x'_a\) is "false". The case is similar if \(u_a\) is negated.

A similar argument holds for \(f_{1j}\), and \(f_{2j}\). Thus, all paths can be routed through, and all cycles are edge-disjoint.

(b) Assume that the Eulerian digraph can be completely decomposed into edge-disjoint circuits having number equal to twice of the number of all main pairs \((12k)\).
Figure 2.19. An Example for \((x_1, x_2', x_3')\)
The \( t_j \) pair can pass through \( G_a, G_b, \) or \( G_c \) straight or slant depending on whether \( u_a, u_b, \) or \( u_c \) is non-negated or negated. We claim that the \( t_j \) pair must pass together through one of \( G_a, G_b, \) and \( G_c \). To see this, assume that the \( t_j \) pair passes separately through \( G_a \) and \( G_b \). Also assume that \( u_a, u_b, \) and \( u_c \) are non-negated. Then the pair \( t_j \), once passed through, will force all other main pairs in \( G_a, G_b \) to pass through by the same way straight. But \( f_{1j} \) and \( f_{2j} \) will need 4 slant paths to go through, and \( G_c \) only provides 2, and this is a contradiction.

Assume that the \( t_j \) pair passes through \( G_a \) straight or slant, then we assign \( u_a \) "true". The corresponding variable is then assigned "false" or "true" according to whether the literal is negated. Assume that the variable is assigned "true". Then all paths for main pairs through \( G_a \) are straight. If there is a pair \( t_j' \) passing through \( G_a \), then \( x_i \) must be "true". If there is a pair \( f_{1j'} \) or \( f_{2j'} \) pass through \( G_a \), then \( x'_i \) must be "false" and this is correct since \( x_i \) is true. Similarly for the case where the variable is assigned "false".

Since the \( t_j \) pair passes through \( G_a \), then \( f_{1j} \) and \( f_{2j} \) must pass through \( G_b \) and \( G_c \). We will assume the literals \( u_b \) and \( u_c \) "false". A similar argument can guarantee that the assignment is consistent for each literal in each clause.
If the \( t_j \) pair passes through \( G_b \) or \( G_c \), then it only introduces notational difficulty. Thus, ONE-IN-THREE is satisfied, and (b) is completed.

From Theorem 2.8, since the literal of a variable occurs at most 5 times, our construction has the number of stages bounded by a fixed number. Q.E.D.

2.6. Parameters \( \tau \) and \( \nu \) in Graphs and Hypergraphs

Given a directed graph \( G=(V,E) \), let \( \tau_{e} \) denote the minimum number of edges meeting (covering, or representing) all circuits in \( G \); that is, there exists a subset \( E' \) of \( E \) such that \( E' \) contains at least one edge from every directed circuit in \( G \), and the cardinality of \( E' \) is minimum. The deletion of \( E' \) will make \( G \) acyclic. Let \( \nu_{e} \) denote the maximum number of edge-disjoint circuits in \( G \), \( \tau_{v} \) denote the minimum number of vertices meeting all circuits in \( G \), and \( \nu_{v} \) denote the maximum number of vertex-disjoint circuits in \( G \). (See [Berg73], [Lova76]).

It has been shown [Karp72] that finding \( \tau_{e} \) (feedback arc set) and finding \( \tau_{v} \) (feedback vertex set) are NP-complete problems. Since VAMIN digraphs and BEMIN digraphs are special cases of general digraphs, and VPPR and PPR are restricted problems of finding \( \nu_{e} \) and finding \( \nu_{v} \) respectively, from Theorem 2.10 and Theorem 2.9 we
have the following result immediately.

**Theorem 2.11:** Given a digraph $G$, it is NP-complete to find $n_u$ or $n_v$ in $G$.

In general digraph, it is known that $n_u \leq \tau_u$ and $n_v \leq \tau_v$. There exist digraphs with $n_u > \tau_u$ and $n_v > \tau_v$. This is also true for Eulerian digraphs. Figure 2.20 shows a digraph with $1 = n_v = n_u < \tau_u = \tau_v = 2$. Figure 2.21 shows an Eulerian digraph with $n_u < \tau_u$ and $n_v < \tau_v$ whose construction is based on the digraph shown in Figure 2.20. It is clear that if an $n \times m$ BEMIN can realize a permutation $P$, then the corresponding BEMIN cyclic digraph has the property $n_u = \tau_u$. Thus a class of $n!$ digraphs with the property $n_u = \tau_u$ can be obtained by using a rearrangeable $n \times m$ BEMIN together with all the $n!$ permutations. Our further interest here concerns whether $n_u = \tau_u$ or $n_v = \tau_v$ for a given digraph.

**Theorem 2.12:** Given a digraph $G$, it is NP-complete to determine if $n_u = \tau_u$ or $n_v = \tau_v$. It is NP-complete even if a set of edges (vertices) meeting all circuits with cardinality $\tau_u$ ($\tau_v$) is given.

**Proof:** We will prove the edge case, and the vertex case is similar. It is only necessary to prove the restricted case stated in the theorem.
Figure 2.20. Simple Digraph with Tau > Nu

Figure 2.21. An Eulerian Digraph with Tau > Nu
Because it is clear that any $n$ edges of a link pattern between any two stages of a $n \times m$ BEMIN digraph will meet all circuits, the following problem is NP-complete by Theorem 2.10.

Problem A: Given a set of $n$ edges meeting all circuits in a digraph $G$, determine if $n_{\text{u}} = n$ in $G$.

The proof will be completed by showing that the problem is equivalent to the problem stated in the theorem. It is only necessary to prove that problem A is reducible to the problem stated in the theorem.

Given an instance $I$ of problem A, we will solve $I$ as follows. If $n < \tau_{\text{u}}$, then the input contains an error since $n > \tau_{\text{u}}$. If $n > \tau_{\text{u}}$, then output "no" since $n > \tau_{\text{u}} > n_{\text{u}}$. If $n = \tau_{\text{u}}$, then use the algorithm for the problem stated in the theorem. Q.E.D.

2.6.1. Generalization to Hypergraphs

A hypergraph $H$ is defined to be a pair $(X, U)$ where $X$ is a set $\{x_1, x_2, \ldots, x_n\}$ of elements called vertices (nodes), and $U$ is a family (or system) $(E_1, E_2, \ldots, E_m)$ of elements, each being a subset of $X$, called edges (arcs).

The NP-completeness results in graphs or digraphs can immediately be generalized to hypergraphs since hypergraphs are the
This generalization does not give us "new" results in essence because graphs and digraphs are in a proper subset of the set of hypergraphs and the generalization only concerns problems with that proper subset. Our generalization here will view digraphs as hypergraphs by the following correspondences.

The vertices of digraphs correspond to the vertices of hypergraphs. The simple circuits of digraphs correspond to the edges of hypergraphs. The edge disjoint (independent) property of edges in hypergraphs is then identified with the node-disjoint property of circuits in digraphs. Thus, \( \tau_v \) is the vertex cover number \( \tau(H) \) and \( \nu_v \) is the matching number \( \nu(H) \) of the hypergraph \( H \).

The NP-completeness results obtained previously for digraphs concerning \( \tau \) and \( \nu \) can then be generalized to hypergraphs for the vertex cover number and the matching number. The above statement being true is based on the condition that the measure of the problem size regarding a given hypergraph is done by the size of its corresponding graph or digraph. That is, the representation of an instance of a hypergraph problem is specified by the instance of the corresponding graph or digraph problem. This generalization may seem "unfair" to one who argues that the problem size regarding a
hypergraph should be measured by the size of the given hypergraph according to its definition. If a given hypergraph is represented by its vertices and edges, the generalization may fail since it involves the problem of generating all circuits in a digraph.

However, even through a given hypergraph is represented by its vertices and edges, the theorems presented previously can still be generalized. Since we have shown that the VPPR problem is NP-complete even if the number of stages $m$ is fixed, the number of simple circuits of length $m$ of the corresponding VAMIN cyclic digraph is bounded and can be generated in polynomial time. Thus, the following applies when we make the correspondence between the minimum length circuits of digraphs and the edges of hypergraphs.

**Theorem 2.13:** It is hard to determine the matching number for a given hypergraph $H=(X,U)$.

**Theorem 2.14:** Given a hypergraph $H=(X,U)$, it is hard to determine if $\nu(H)=\tau(H)$. It is hard even if a minimum set of vertices covering all edges is given.
2.7. Summary

By limiting the switching elements in an interconnection network to be exclusively binary elements, we study the effect of the link patterns on the permutations the network can realize. The particular permutation realization problem in connection with the design of multistage interconnection networks is presented to reflect the difficulty of analyzing the permutations a BEMIN can realize once all its link patterns are specified. We present some polynomially decidable cases of PPR in some restricted situations.

The PPR problem is transformed to some polynomially equivalent problems. It is then found that its vertex-version (VPPR) is an unsolved subproblem of the subgraph homeomorphism problem.

We settle both PPR and VPPR to be NP-complete. The proof of the NP-completeness of PPR and VPPR is based on the reduction from the ONE-IN-THREE 3SAT problem, and we expect that the method would be applied to other cases. The results are applied to obtain more NP-complete results related to problems regarding two graph theoretic parameters, tau and nu. All these justify the existence of the class of NP-complete problems which is deemed by most computer scientists to be not solvable by any polynomial-time algorithm.
The reflection on multistage interconnection network design is that in theory it is not likely to analyze any network for its realizable permutations efficiently. Thus, this is a clue which shows that one cannot check the rearrangeability any given BEMIN in an efficient systematic way.
EFFICIENT MEMBERSHIP-RECOGNITION AND PERMUTATION-RECOGNITION CLASSES OF MULTISTAGE INTERCONNECTION NETWORKS

The NP-completeness of the PPR problem tells us that the general study of link patterns is a hard problem. However, we can limit ourselves to certain specific types of link patterns and analyze them efficiently. What relations can we introduce among link patterns such that they will result networks which can be analyzed efficiently? This chapter answers this question by giving some interesting classes of networks. Of course, the usefulness of certain link patterns does not guarantee us that the other link patterns are useless.

3.1. Classifying Multistage Interconnection Networks by the Particular Permutation Realization Problem

In Chapter 1, an n×m BEMIN (n by n, m stage binary element multistage interconnection network) is defined as a pairwise-connected one-way two-sided multistage interconnection network which has n inlets and n outlets, n being even, m stages of switching elements, each using binary elements exclusively, and m+1 link patterns, F₀, F₁, ..., Fₘ. See Figure 1.8. A binary element is a
primitive 2*1 BEMIN with $F_0$ and $F_1$ being the identity permutation and it is rearrangeable. See Figure 1.7 for the only two possible states.

A state of an interconnection network is also defined in Chapter 1. A state of an $n \times m$ BEMIN is a physical realizable or admissible one-one connection from all inlets to all outlets. There are two ways to specify states of a BEMIN by permutations. Let both inlets and outlets be labeled by the elements of an ordered set, say $1, 2, \ldots, n$, from top to bottom. A state can be specified as a permutation of the inlets at the outlets or as a permutation of the outlets at the inlets. When a BEMIN is used for permutation purposes such as in a data manipulator [Feng74], the former approach is usually used, because the situation requires that a set of data at the inlets be permuted according to a permutation and the result occur at the outlets. When a BEMIN is used for message passing purposes, the latter approach is usually used. In this case, each inlets hold a message to be sent to a distinct outlet according to a permutation which may be the permutation of the outlet addresses.

An $n \times m$ BEMIN can be specified as $F_0 S_1 F_1 S_2 \ldots F_{m-1} S_m F_m$, where $F_i$ is a link pattern denoted by a permutation of degree $n$, and $S_i$ is a stage of $n/2$ binary elements. When the states of all binary elements of a stage $S_i$ are set, the stage also realizes a
permutation $E_1$ of degree $n$ which may interchange some pairs of terminals connected to the same binary element. Let a permutation $P$ be a state of an $n \times m$ BEMIN. The specification of the permutations of the link patterns and the states of the stages will be consistent with that of the state; that is, either they will all be the permutations of the inlet-side terminals at the outlet-side terminals or all be the permutations of the outlet-side terminals at the inlet-side terminals. The realization of $P$ can then be viewed as a composition (or product) of a sequence of permutations assumed by the link patterns and the states of the stages. We write $F_0E_1F_1E_2...F_{m-1}E_mE_m=P$. If the former convention is used, the sequence will be a composition operated from right to left. If the latter convention is used, the sequence will be a product operated from left to right. It is clear that the two approaches are consistent since $P^{-1} = (F_0E_1F_1E_2...F_{m-1}E_mE_m)^{-1} = F_{m-1}E_{m-1}^{-1}F_{m-1}^{-1}...E_2^{-1} F_1^{-1}E_1^{-1}F_0^{-1}$. From another point of view, in the first approach an $n \times m$ BEMIN "generates" a subset of the set of the symmetric group $S_n$, while in the second approach it "recognizes" or "accepts" a subset of $S_n$.

The PPR (particular permutation realization) problem has also been presented as the problem of determining if a given $n \times m$ BEMIN can realize a given permutation. Can this problem be solved efficiently? Since a BEMIN realizes a set of permutations, we
actually ask whether it is efficient to do membership test for a given permutation against the set of permutations realized by the given BEMIN. We have shown that PPR is an NP-complete problem in Chapter 2. This implies that the whole class of BEMINs contains some "complex" networks with respect to the PPR problem. On the other hand, we have also shown that there are some restricted classes of networks with the corresponding PPR problems being polynomially solvable. That is, some classes of BEMINs are "simple" in the sense that there is an algorithm which can solve whether a given permutation can be realized by a given BEMIN in these classes in some "reasonably" efficient time. Assuming that the class of polynomially solvable problems by deterministic algorithms (P) is not equal to the class of polynomially solvable problems by nondeterministic algorithms (NP), the PPR problem "divides" the whole class of BEMINs into two parts: a collection of BEMINs whose PPR problems are in P, and a collection of BEMINs whose PPR problems are in NP and not in P.

If a class of BEMINs whose PPR problem is polynomially solvable, we call the class to be PPR-polynomial or polynomial-time permutation-recognizable. If the PPR problem can be solved in \( O(p(x)) \), \( p(x) \) being polynomial, time, we say that the corresponding class of BEMINs is PPR-polynomial or polynomial-time permutation-recognizable in \( O(p(x)) \). A class of BEMINs is robust if
there exists a polynomial-time algorithm to check whether a given BEMIN is in the class. That is, the test of membership of a given arbitrary BEMIN against the class can be done efficiently, and the class of BEMIN has the property of polynomial-time membership-recognition. Here we are interested in the robust classes of BEMINs with PPR-polynomial property. It has been shown in Chapter 2 that the following classes of BEMINs are PPR-polynomial, and it is clear that these classes are robust.

1. The class of \( n \times m \) BEMINs where \( n \) is a certain constant.
2. The class of 2-stage BEMINs.
3. The class of 3-stage BEMINs.

In this chapter, we will find some more interesting classes of PPR-polynomial BEMINs. We will treat "BEMIN" and "network" as synonymous terms. We say easy when we mean polynomial-time, and we say hard when NP-complete or NP-hard.

### 3.2. The Reverse Network

Let \( \alpha \) be an \( n \times m \) BEMIN, the reverse network of \( \alpha \), \( \alpha^{-1} \), is the \( n \times m \) BEMIN obtained by switching the input and the output side of \( \alpha \). Clearly, \( \alpha \) is the reverse network of \( \alpha^{-1} \). Let \( \alpha \) be specified as \( F_0S_1F_1S_2...F_{m-1}S_mF_m \).

If \( \alpha \) realizes a permutation \( P \), \( F_0E_1F_1E_2...F_{m-1}E_mF_m = P \).

Then \( \alpha^{-1} \) realizes \( P^{-1} \),
since $P^{-1} = F_{m-1}E_{m-1}F_{m-1}^{-1}\ldots E_2^{-1} F_1^{-1} E_1^{-1} F_0^{-1}$,

and $\alpha^{-1}$ is specified by $F'_0S_1F'_1S_2\ldots F'_{m-1}S_mF'_m$,

where $F'_0 = F_{m-1}^{-1}$, $F'_1 = F_{m-1}^{-1}$, $\ldots$, $F'_{m-1} = F_1^{-1}$, and $F'_m = F_0^{-1}$.

Obviously, a network and its reverse network realize the same number of permutations.

Theorem 3.1: If $\{X\}$ is a PPR-polynomial class of networks and $\{X^{-1}\}$ is the class of networks containing only the reverse networks of the networks in $\{X\}$. Then $\{X\} \cup \{X^{-1}\}$ is PPR-polynomial. If $\{X\}$ is robust, then $\{X\} \cup \{X^{-1}\}$ is robust.

Proof: It is easy to get the reverse network of a given network. The problem of deciding if $\alpha$ can realize $P$ is equivalent to the problem of deciding if $\alpha^{-1}$ can realize $P^{-1}$. It is easy to solve at least one of the two problems. For robustness, given a network $\alpha$, $\alpha$ is in $\{X\} \cup \{X^{-1}\}$ if and only if either $\alpha$ or $\alpha^{-1}$ is in $\{X\}$.

Example 3.1: It follows directly from the definitions of the networks that the following networks in each pair are reverse to each other.

1. The reverse-exchange (or the baseline) network and the reverse baseline network [Feng79].
2. The omega network [Lawr75] and the flip network [Batc76].

3. The modified data manipulator [Feng74] and the banyan (S=F=2) network [Goke73] (or the indirect binary n-cube network [Peas77]).

3.3. Topological Equivalence

Two n*m BEMINs are topologically equivalent iff their corresponding BEMIN acyclic digraphs (see Chapter 2) are isomorphic.

Two link patterns are topologically equivalent iff the 2-stage BEMINs associated by the link patterns are topologically equivalent.

Theorem 3.2: It is easy to decide if two link patterns are topologically equivalent.

Proof: The link patterns can be classified by 2*k-cycles, k=1, 2,..., as shown in Figure 2.7. We test for the number of each existing alternating 2*k-cycles for the two link patterns. Q.E.D.

Theorem 3.3: If two networks are topologically equivalent, all the corresponding link patterns must be topologically equivalent.

Proof: All the arcs of a BEMIN acyclic digraph are directed from the inlet-side to the outlet-side. Thus, the isomorphism must be "stage-wise". Q.E.D.

Clearly, even if all corresponding link patterns are topologically equivalent for two given BEMINs, the BEMINs may not be
topologically equivalent. All the networks listed in Example 3.1 are topologically equivalent [Feng79]. An observation shows that they all have 4-cycles property for all link patterns.

We present the following functional transformation for two topologically equivalent networks.

**Theorem 3.4:** If two BEMINs $\alpha$ and $\beta$ are topologically equivalent and $\alpha$ can realize a permutation $P$, then $\beta$ can also realize $P$ provided that two fixed (independent of $P$) patterns, an input pattern $I$ and an output pattern $O$, are added to $\beta$.

**Proof:** Let $\alpha$ and $\beta$ be two $n \times m$ BEMINs. See Figure 3.1 for illustration. We also use $\alpha=(V_1,A_1)$ and $\beta=(V_2,A_2)$ to denote the corresponding BEMIN acyclic digraphs. Let each binary element be denoted by $b_{ij}$ if it is in the $i$th level from top to bottom and in the $j$th stage from left to right.

Since $\alpha$ and $\beta$ are topologically equivalent, there is a bijection $f:V_1 \rightarrow V_2$ such that the link $(u,v)$ in $A_1$ iff the link $(f(u),f(v))$ in $A_2$. Furthermore, from Theorem 3.3, the bijection can be partitioned into $m+2$ bijections, $f_i$, $i=0,1,\ldots,m+1$, $f_0$ for the $n$ input nodes, $f_{m+1}$ for the $n$ output nodes, and $f_j$, $j=1,2,\ldots,m$, for a stage of $n/2$ nodes. Let the domains of all bijection be $\{0,1,2\ldots\}$. 
Figure 3.1. Functional Equivalence and Topological Equivalence
Define I to be the permutation $(f_0(0), f_0(1), \ldots, f_0(n-1))$ and $0$ to be the inverse of the permutation $(f_{m+1}(0), f_{m+1}(1), \ldots, f_{m+1}(n-1))$.

Let $[C]$ be an $(n/2) \times m$ bit-matrix for alpha to realize the permutation $P=(p_0, p_1, \ldots, p_{n-1})$. That is, $c_{ij}=x$ iff the binary element $b_{ij}$ is at state $x$. Then there are $n$ disjoint paths which connect the output node $j$ to the input node $p_j$, $0 \leq j \leq n-1$.

We write $[C]|_{alpha} = \Rightarrow P$. The operation $[C]$ on alpha can be viewed as the composition of a sequence of permutations defined by the link patterns and the states of the stages of the network from the output side to the input side.

Let the labeling of the network alpha be done from top to bottom for the input nodes, $0, 1, \ldots, n$, for the output nodes, $0, 1, \ldots, n$, for the binary elements of each stage $0, 1, \ldots, n/2$, and for the links connected to a binary element, $(v_1, v_{tl})$, $(v_2, v_{bl})$, $(v_3, v_{tr})$, and $(v_4, v_{br})$, where $v_{tl}=v_{bl}=v_{tr}=v_{br}$ is the label of the binary element, and $v_1, v_2, v_3$, and $v_4$ are the neighboring nodes connected to the binary element through the top-left, bottom-left, top-right, and bottom-right link respectively. The network beta is then labeled by the image of the
mapping $f$ which assumes the topological equivalence. We define the sign of a binary element by the labeling of the links connected to it.

Let $\text{sign}(b_{ij}) = \frac{(v_2-v_1)(v_4-v_3)}{|(v_2-v_1)(v_4-v_3)|}$.\(^1\)

Let $[C']$ be a bit-matrix for beta such that

$c'_{ij} = y$ iff $f_j^{-1}(b_{ij})$ is at state $x$,
where $y$ is either $x$ or the complement of $x$ depending on the labeling of the links connected to the binary element.

We set $y = x$ iff $\text{sign}(b_{ij}) = \text{sign}(f_j^{-1}(b_{ij}))$.

Then there are $n$ disjoint paths which connect the output node $f_{m+1}(j)$ to the input node $f_0(P_{m+1}(j))$, $0 \leq j \leq n-1$.

Clearly, $I[C']\beta\alpha_0 \Rightarrow P$. Since the above is true for any permutation which is realizable by alpha (one can consider a permutation $Q$ realizable by alpha which is a transposition of $P$), the proof is completed. Q.E.D.

The beta network plus $I$ and $0$ defines another network gamma. Obviously, $[C']\gamma\alpha_0 \Rightarrow P$ as a continuation from the proof. Also, we have $[C']\beta\alpha_0 \Rightarrow I^{-1}P_0^{-1} = P'$, and the mapping from $P$ to $P'$ is a

\(^1\) $|x|$ denotes the absolute value of $x$. 
bijection since 1 and 0 are fixed permutations. We exemplify the above as follows.

Example 3.2: Figure 3.2(a) is an 8*3 BEMIN (alpha) which is a baseline network. The realization of a permutation $P$ is shown. Thus, $[C]\alpha \Rightarrow P$ is

$$
\begin{array}{c|c|c|c|c|c|c|c}
\alpha & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{array}

\Rightarrow (3 \ 7 \ 0 \ 4 \ 5 \ 2 \ 1 \ 6).
$$

In Figure 3.2(b) there is another 8*3 BEMIN (beta) which is similar to an omega network. The topological equivalence is shown by the labeling of the nodes of beta according to the image of the isomorphic mapping from alpha to beta. The nodes labeled are the input nodes, the output nodes, and the nodes for binary elements. Fixed patterns 1 and 0 are then defined as shown in Figure 3.2(b). The links can then be labeled by the labeling of the nodes, and the bit-matrix $[C']$ is set accordingly. Note that the sign of the second binary element in stage 1 of beta is negative and all other binary elements of alpha and beta are positive. Then $I[C']\beta \Rightarrow P$, that is

$$
\begin{array}{c|c|c|c|c|c|c|c}
\beta & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
\hline
1 & 4 & 5 & 2 & 3 & 6 & 7 & 1 \\
\end{array}

\Rightarrow (3 \ 7 \ 0 \ 4 \ 5 \ 2 \ 1 \ 6),
$$
Figure 3.2. An Example of Functional Transformation

(a) (to be continued)
and \([C']_{\text{beta}} \Rightarrow \overline{I-1}P_0-1=I', \text{i.e.}\]

\[
\begin{array}{c|ccc}
  & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
1 & 1 & 1 \\
2 & 1 & 0 & 0 \\
\end{array}
\Rightarrow (5 \ 3 \ 7 \ 4 \ 0 \ 1 \ 2 \ 6).
\]

We also have \([C']_{\text{gamma}} \Rightarrow P.\]

We calculate that \(I'P_0=P\) as follows.

\[
\begin{align*}
I'P_0(X) &= IP'0(\begin{array}{c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}) \\
&= I(\begin{array}{c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} / (\begin{array}{c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array})) \\
&= I(\begin{array}{c}
0 & 1 & 4 & 5 & 2 & 3 & 6 & 7 \\
\end{array} / (\begin{array}{c}
5 & 3 & 7 & 4 & 0 & 1 & 2 & 6 \\
\end{array})) \\
&= \begin{array}{c}
3 & 7 & 0 & 4 & 5 & 2 & 1 & 6 \\
\end{array}/(X) \\
&= P(X),
\end{align*}
\]

where \(X\) is any permutation.

**Corollary 3.5:** Two topologically equivalent \(n^*m\) BEMINs realize the same number of permutations.

Topologically equivalence is a sufficient condition for a network to be included in a PPR-polynomial class of BEMINs. Thus, the PPR-polynomial class of BEMINs can be expanded in this sense.

**Corollary 3.6:** If \(\{X\}\) is a PPR-polynomial class of networks and
a network beta is topologically equivalent to a BEMIN alpha which is in \{X\}, then \{X\}∪{beta} is PPR-polynomial provided that alpha can be found in polynomial time in \{X\} for beta, and the mapping for alpha and beta can be done efficiently such that 1 and 0 can be decided in polynomial time.

A problem involved here is whether it is easy to determine that two BEMINs are topologically equivalent [Fure83].

3.4. Transversal Networks and Partial Transversal Networks

We will restrict our attention to n×m BEMINs where n=2N, N a natural number. A basic theorem under transversal networks is a theorem by P. Hall [Hall35], we review it here as follows.

Let E and I be sets; let g:I→I be a mapping and write g(i)=x_i for all i in I. Then the mapping g denoted by the symbol \(x_i:i \text{ in } I\) is called a family (or system) of elements of E indexed by I (or with index set I).

Let \(U=(A_i:i \text{ in } I)\) be a family of subsets of a ground set E. A subset T of E is a transversal (the range of the system of distinct representatives) of \(U\) iff there exists a bijection \(f:T→I\) such that \(x \text{ in } A_{f(x)}\) for all \(x \text{ in } T\) [Mirs71].

Theorem 3.7: The finite family \(U=(A_i:1 ≤ i ≤ n)\) of subsets of a set
E possesses a transversal iff it satisfies Hall's condition, i.e. iff for each I contained in \{1, \ldots, n\}, |A(I)| \geq |I|, where A(I) is the union of (A_i: i \text{ in } I).

Hall's condition states that the union of any k sets has at least k elements and Hall's theorem states that Hall's condition is necessary and sufficient for a finite family of subsets to have a system of distinct representatives.

3.4.1. The General and the Standard Benes Binary Network

A general construction scheme for a class of rearrangeable has been given by [Bene65]. We restrict the attention to networks built by binary elements and the number of inlets/outlets is \( n=2^N \).

The recursive construction or definition of the general Benes binary networks (GBBN) is given as follows.

(Base) A 1-stage GBBN (of size 2) is any primitive 2*1 BEMIN, i.e. an arbitrary permutation \( F_L \) of degree 2, followed by a binary element, and followed by another arbitrary permutation \( F_R \). See Figure 3.3.

(Recursion) A \((2N-1)\)-stage GBBN (of size \( n \)) is an \( n*(2(\log(n))-1) \) BEMIN with link patterns defined by the following recursive construction rule. A \((2N-1)\)-stage GBBN consists of an
Figure 3.3. A 1-Stage GBBN

Figure 3.4. The Recursive Construction of GBBN
arbitrary permutation pattern $F_L$ of degree $n$, followed by a stage of $n/2$ binary elements, followed by an unshuffle (see Chapter 4) pattern of degree $n$, cascaded by two $(2N-3)$-stage GBBNs, a top one GBBN$_T$ and a bottom one GBBN$_B$, then followed by a shuffle pattern of degree $n$, followed by a stage of $n/2$ binary elements, and followed by an arbitrary pattern $F_R$.

Figure 3.4 shows the first level recursion for a GBBN. The general case of the rearrangeability of Benes networks are proved in [Bene65] where the Slepian-Duguid theorem is cited. We omit the proof for the GBBN case here, since it is a special case and can be found elsewhere [Waks68].

**Theorem 3.8**: Any GBBN is rearrangeable.

**Corollary 3.9**: The class of GBBNs is PPR-polynomial.

A standard Benes binary network (BBN) is a GBBN where all arbitrary permutations during the construction process are the identity permutation. Figure 3.5 shows the first and the second level of recursion for construction of a BBN. The $2^r$ BBN's created in the recursion level $r$ are labeled by $BBN_i$ from top to bottom, where $i$ is the binary representation and $0 < i < 2^r - 1$. A 5 stage BBN (or a BBN of size 8) is given in Figure 1.9.
Figure 3.5. The Recursive Construction of BBN
3.4.2. Partial Transversal Networks

A transversal network (TN) is an n*m BEMIN which is topologically equivalent to a GBBN of size n. Two transversal networks are transversally equivalent, but they may not be topologically equivalent.

Corollary 3.10: Any transversal network is rearrangeable and the class of transversal networks is PPR-polynomial.

An m-stage partial transversal network (PTN) of size n is an n*m BEMIN which is obtained by taking the first m stages of a transversal network of size n. The partial GBBN and partial BBN are similarly defined. The reverse network of a PTN is denoted by PTN⁻¹.

Theorem 3.11: The number of permutations realizable by an m-stage PTN of size n is

\[ 2^{mn/2} \text{ when } m \leq \log(n), \text{ and } \]

\[ 2^{(n/2)(2(\log(n))-1-m)}[(2^m+1/n)!]n^{2/2^m+1} \text{ when } m > \log(n). \]

Proof: From Corollary 3.5, we need only to prove for the partial GBBNs.

(a) The case \( m \leq \log(n) \).

Each \((n/2)^m \) bit-matrix \([C]\) corresponds to a state of the n*m BEMIN
and no two \([C]\)'s realize the same state. Thus, the number of \([C]\)'s, \(2^{mn}/2\), implies the number of realizable permutations.

(b) The case \(m \geq \log(n)\).

The last \(y = 2(m - \log(n)) + 1\) stages form \(n/z\) rearrangeable networks, from top to bottom, each having size \(z = 2(y+1)/2\). The first \(x = m - y\) stages have the situation of case (a). Thus, each permutation realized by the first \(x\) stages can be partitioned into \(z\) parts from top to bottom, and each part can generate all possible permutations in the last \(y\) stages. See Figure 3.6 for illustration. Hence, we have the formula given in the theorem.

Note that the two formulas give the same result for the case \(m = \log(n)\). Q.E.D.

Example 3.3: For transversal networks of size 8, Table 3.1 gives the number of permutations realizable by the PTNs.

Lemma 3.12: The class of partial GBBNs is PPR-polynomial.

Proof: Let the partial GBBN be of size \(n\) and have \(m\) stages. Assuming that the last link pattern \(F_m\) is the identity does not lose generality.

If \(m < \log(n)\), for solving the corresponding PPR, it follows from the fact that there is a single path for any inlet to be connected
Figure 3.6. Partial GBBN Scheme for $m > \log(n)$

Table 3.1. Number of Permutations of PTNs of Size 8

<table>
<thead>
<tr>
<th># of stages</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td># of permutations</td>
<td>16</td>
<td>256</td>
<td>4096</td>
<td>9216</td>
<td>40320</td>
</tr>
</tbody>
</table>
For the case \( m > \log(n) \), we divide the network into two parts. The first part contains the first \( x \) stages and the second part contains the last \( y \) stages, \( x + y = m \), as what is done in case (b) of the proof of Theorem 3.11. Thus, the second part contains \( n/z \) rearrangeable networks from top to bottom, each being of size \( z \). Let the input link pattern \( F_0 \) of all the \( n/z \) rearrangeable networks be the identity.

Suppose \( P = (p_1, p_2, \ldots, p_n) \) is the permutation to be tested for realizability of the PPR problem. Let sources (input terminals) be labeled 1, 2, \ldots, \( n \) from top to bottom. Let destinations (output terminals) be labeled according to \( P \) from top to bottom.

Define \( P^* = \left( \{p_1, p_2, \ldots, p_z\}, \{p_{z+1}, \ldots, p_{2z}\}, \ldots, \{p_{n-z+1}, p_n\} \right) \) to be the set of permutations such that if any member can be realized by the first part of the network, then \( P \) can be realized by the given partial GBBN. Thus, a bunch \( B_i = \{p_{(i-1)z+1}, \ldots, p_{iz}\} \) represents any permutation of \( p_{(i-1)z+1}, \ldots, p_{iz} \). The terminals for the bunches are also labeled from top to bottom. The following algorithm tests if the first part of the network can realize \( P^* \).

**Step 1.** Let \( k = 1 \).

**Step 2.** Find \( B_i \) such that destination \( k \) is in \( B_i \).

**Step 3.** If no path from source \( k \) to any destination in \( B_i \), then
output "no" and stop.
Step 4. If \( k = n \), then output "yes" and stop.
Step 5. Delete the path from source \( k \) to destination \( k \) which is found in \( B_1 \) in step 3 and fixed now. Increment \( k \) by 1 and go to step 2.

Apparently, the corresponding PPR problem is solved in polynomial time.

To prove that the algorithm is sufficient, we claim that there is only one possible path for any source \( k \) to be connected to a bunch \( B_i \). Let each bunch be a "bunch" node formed by identified all nodes in the bunch. An observation immediately shows that all the paths from the source node which source \( k \) is connected to, to the bunch nodes \( B_i, i=1,\ldots,b \), form a tree with the source node being the root and bunch nodes being the leaves. Thus the execution of step 5 in the algorithm only deletes a path with single possible way to form a connection, and the proof is completed. Q.E.D.

Lemma 3.13: The class of partial transversal networks is robust.

Proof: From the definition of GBBN, we see that given an \( n \times m \) BMIN, it is easy to decide whether it is a partial GBBN. Let any partial GBBN be labeled such that a binary element is labeled \( b_{ij} \) if
it is in stage \( j \) and at level \( i \) from the top. By definition, the binary elements of a BEMIN alpha can be labeled according to a partial GBBN iff alpha is a PTN. The following algorithm gives an isomorphic (canonical) labeling for an \( n \times m \) BEMIN if it is a PTN. We will assume that \( 2(\log(n)) - 1 > m > \log(n) \). If \( m < \log(n) \), one can just start from the proper place of the procedure.

The labeling process is first performed from the \( \log(m) \) stage to the input and output sides. Let a set contain the nodes which are in the same GBBN at a certain level. Initially, there are \( n/2 \) sets, each one containing one binary element. While the process moves to the two sides, the sets are enlarged since the GBBNs are expanded.

At each step, include one adjacent stage in either direction, if there is one. In the mean time, put the binary elements into correct sets by examining the link patterns in a top to bottom fashion such that the sets of binary elements are "unioned" into the same set if they should belong to the same enlarged GBBN. The examinations are successful iff once a binary element joins the two sets, then any node with a link connected to one set must have the other link connected to the other set. The process can reach the two sides iff the network is a PTN to which a GBBN is corresponding.

Then, from the two sides moving into the stage \( \log(n) \), the
network can be labeled according to the labeling of the corresponding GBBN in a top to bottom fashion. It is clear that the labeling algorithm runs in polynomial time. Q.E.D.

Lemma 3.14: The class of partial transversal networks is PPR-polynomial.

Proof: The proof is a modification of Lemma 3.12 such that we can form the correct bunches from the second part of the network. From the canonical labeling given in the proof of Lemma 3.13, we are able to form the corresponding bunches by stepping from the middle stage of the rearrangeable networks of the second part to the two sides of the rearrangeable networks. The permutation at the outlets can then be mapped into bunches for the test of the first part. Q.E.D.

Theorem 3.15: The class of partial transversal networks and their reverse networks, \( \{PTN\} \cup \{PTN^{-1}\} \), is PPR-polynomial and it is robust.

Proof: The PPR-polynomial property follows from Theorem 3.1 and Lemma 3.14, and the robust property follows from Theorem 3.1 and Lemma 3.13. Q.E.D.
3.5. Cutting Transversal Networks in the Middle

When arbitrary TNs are cut in the middle, many interesting networks can be identified with different properties. In this section, we discuss the classes of left-half, right-half, left-right, and half transversal networks together with a class of networks possessing a simple distributed control scheme.

3.5.1. Conjugate Networks

For the discussion in this subsection, we assume that the output link pattern (the pattern $F_m$) of an $m$-stage PTN is changed to be identity permutation for convenience.

A left-half transversal network (LT) is the log(n)-stage PTN of a transversal network of size n. A right-half transversal network (RT) is the reverse network of some LT.

The left-half and the right-half of GBBN and BBN are similarly defined. From this definition, the reverse-exchange (or the baseline) network is the left-half of BBN.

An LT alpha and an RT beta are conjugate iff the network, $(alpha)(beta)$, formed by cascading alpha with beta and merging the last stage of alpha and the first stage of beta is a TN. We call alpha the left conjugate of beta, denoting alpha=*$beta, and beta the
right conjugate of alpha, denoting beta=alpha*.

Since (alpha)(alpha)-1 and (beta)-1(beta) are rearrangeable, we have the following.

Theorem 3.16: If alpha is an LT, then (alpha)-1 is a right conjugate of alpha. If beta is an RT, then (beta)-1 is a left conjugate of beta.

The above theorem gives an immediate conjugate network for an LT or an RT; hence, a rearrangeable network can be formed. It is understood that this is not the only way to form transversal, and hence rearrangeable, networks.

Lemma 3.17: Let an LT alpha be labeled according to a topologically equivalent left-half GBBN. An RT beta is a conjugate of alpha iff beta can be labeled according to a topologically equivalent right-half GBBN such that each first stage binary element of beta has the same label as the corresponding last stage binary element of alpha from top to bottom.

Proof: Since any left-half GBBN cascaded and merged by any right-half GBBN of the same size forms a rearrangeable network, (alpha)(beta) is rearrangeable by the topological equivalence. Q.E.D.
The duality of Lemma 3.17 can be stated as follows. Let an RT beta be labeled according to a topologically equivalent right-half GBBN. An LT alpha is a conjugate of beta if alpha can be labeled according to a topologically equivalent left-half GBBN such that each first stage binary element of beta has the same label as the corresponding last stage binary element of alpha from top to bottom.

Define the **stripped network** of a network by changing the first \((F_0)\) and the last \((F_m)\) link patterns of the network to the identity permutation.

Many rearrangeable networks can be obtained by applying the lemma and its duality above. An example is given as follows.

Example 3.4: By canonical labeling, we have the following.

The right conjugates of the baseline, the reverse baseline, the modified data manipulator, or the omega network are

1. the banyan network.
2. the flip network.
3. the reverse baseline network.
4. the baseline network.

The left conjugates of the baseline, the reverse baseline, the banyan, or the flip network are

1. the modified data manipulator.
2. the omega network.
3. the reverse baseline network.
4. the baseline network.

The right (left) conjugates of the banyan or the stripped flip (the modified data manipulator, the stripped omega) network are
1. the modified data manipulator (the banyan) network.
2. the stripped omega (the stripped flip, respectively) network.

If an LT and an RT of the same size are not conjugate, it is not known whether they form a transversal network by cascading the two networks. However, if we introduce a pattern M as the input pattern of the RT which makes the RT act as a conjugate of the LT from Lemma 3.17, we can indeed form a transversal network.

Theorem 3.18: Given an LT alpha and an RT beta, both of size n, there exists a link pattern M being a permutation of degree n such that the network (alpha)(M)(beta) which is formed by merging the last stage of alpha and the first stage of beta due to the introduction of M, is a transversal network.

If alpha and beta are conjugate, M is the identity permutation in the above theorem. Also, the network formed by cascading alpha, M, and beta is rearrangeable.

As an example, (omega)(rho)(omega), (flip)(rho)(flip), (banyan)(delta)(banyan), etc., are all rearrangeable networks, and
the bit-reversal permutation (\( \rho \)) is defined as
\[
\rho([p_{m}p_{m-1}\ldots p_{1}])=[p_{1}\ldots p_{m-1}p_{m}]
\]
for degree \( n=2^{m} \), where \( p=[p_{m}p_{m-1}\ldots p_{1}] \) is the binary representation of any point of the permutation of degree \( n \), and \( \Delta \) is defined as the product of \( \rho \) and \( \text{sh} \).

A PTN being both an LT and an RT, when stripped, is called a left/right transversal network (LRT).

Lemma 3.19: The link patterns of a transversal network of size \( n \) contains only \( 2^{k} \)-cycles where \( k \) must be even. Furthermore, for any link pattern \( F_{i}, 1 \leq i \leq m-1 \), all \( 2^{k} \)-cycles must satisfy \( k \leq 2^{m-1} \), and for any link pattern \( F_{m+i-1}, 1 \leq i \leq m-1 \), all \( 2^{k} \)-cycles must satisfy \( k \leq 2i \), where \( m=\log(n) \).

Proof: In the direction from the inlets/outlets side of the network to the middle, any node of a stage must be led to one "upper" node and one "lower" node in the next stage by the two links connected to it. The upper node corresponds to a node in the \( \text{GBBN}_{T} \) and the lower node corresponds to a node in the \( \text{GBBN}_{B} \). For all node belonging to the same \( \text{GBBN} \) in a stage, they must be led to the same two \( \text{GBBN}_{T} \) and \( \text{GBBN}_{B} \) in the next stage. We call the above process the dichotomizing or dividing process.

If \( k \) is odd, the nodes in a stage of the \( 2^{k} \)-cycles cannot be
dichotomized into 2 disjoint subsets, one containing all upper nodes and the other containing all lower nodes, of the same size from the other stage of the \(2^k\)-cycles in the proper direction stated above.

The maximum cycle length conditions for link patterns \(F_i, 1 \leq i \leq 2(\log(n)) - 2\), follow from the recursive definition of GBBN and Theorem 3.3. Q.E.D.

From Lemma 3.19, we know that \(k\) is even for all \(2^k\)-cycles, and from both the inlet and the outlet sides toward center the maximum \(k\) must be 2, 4, 8, ... etc. Now consider an LRT which has the common property of an LT and an RT. An examination from the dichotomizing process done in Lemma 3.19 shows that \(k\) must be 2 if the dichotomizing process can be done in both directions. Since all link patterns are topologically equivalent with length 4 for all cycles, the canonical labeling assures that any two LRTs are topologically equivalent. Thus, we obtain the following.

Theorem 3.20: An LRT has the property that all the link patterns contain alternating 4-cycles. All LRTs are topologically equivalent.

All the networks given above in the example are LRTs. A characterization of the reverse omega network has been worked out in [Peas77]. Let \(x = [x_m, x_{m-1}, \ldots, x_1]\) and \(y = [y_m, y_{m-1}, \ldots, y_1]\), \(m = \log(n)\),
be the indices in binary notation of some inlet and some outlet, respectively. The set of permutations realized by \( \omega^{-1} \) of size \( n \) is 
\[
\{ P(x) \| y_i = x_i \oplus f_1(y_1, \ldots, y_{i-1}, x_{i+1}, \ldots, x_m) \}
\]
where the \( f_1 \)'s are arbitrary Boolean functions of \( m-1 \) variables and \( \oplus \) denotes exclusive-or or arithmetic addition modulo 2. Thus, there are \( 2^{mn/2} \) different permutations realized by \( \omega^{-1} \) of size \( n \) and defined by the set of \( m \) Boolean equations above.

In [Lee81], this characterization is interpreted as follows. A permutation \( P = (p_0, p_1, \ldots, p_{n-1}) \) is \( \omega^{-1} \) realizable iff \( p_j,k \) forms a complete residue system modulo \( 2^j \), \( 1 \leq j < n \), where \( p_j,k \) is 
\[
(p_{k2^j}, p_{k2^j+1}, \ldots, p_{(k+1)2^j-1}), 0 \leq k < 2^{n-1}.
\]
Since \( \omega^{-1} \) is a LRT, the good characterization is applicable to any LRT \( \alpha \) once the fixed patterns \( I \) and \( O \) for topological equivalence are found. By canonical labeling, \( I \) and \( O \) can be found in polynomial time. To test if permutation \( P \) is \( \alpha \) realizable, one can simply test if the permutation product \( IPO \) is \( \omega^{-1} \) realizable. Thus, a good characterization [Lova79] is obtained for the class of LRTs.

**Theorem 3.21:** A network \( \alpha \) is an LRT iff there exists two permutations \( I \) and \( O \) such that the set of permutation realizable by \( I(\alpha)O \) satisfies the \( \omega^{-1} \)-realizable condition.

**Example 3.5:** We create a new type of networks by using the bit-reversal permutation for link patterns between stages. The
permutation rho has the property that it scatters adjacent members approximately evenly over the interval \(0 \leq x < n\). We define a scatter network of size \(n = 2^m\) as an \(n \times m\) BEMIN with link pattern \(F_i\) contains \(i\) rho's from top to bottom, \(1 \leq i \leq m\), and \(F_0\) is the identity permutation. Figure 3.7(a) contains a scatter network of size 16. Figure 3.7(b) contains an omega^{-1} network of size 16. Both networks are canonically labeled. The scatter network is made to be functionally equivalent to the omega^{-1} network by adding the link patterns, \(I\) and \(O\). It is not difficult to check that they are topologically equivalent and the two fixed patterns \(I\) and \(O\) can be found easily as shown in Figure 3.7(a). Thus, the scatter networks are LRTs.

Next we consider some LRTs which is more powerful in the sense that they preserve the rearrangeability "in half". An LT (or an RT) alpha is a half transversal network (HT) iff \((\alpha)(\alpha)\) forms a transversal network, by merging the middle two stages. For example, the baseline and the reverse baseline network are HTs.

**Corollary 3.22:** A stripped LT (or RT) alpha is an HT iff it is an LRT and the labeling of alpha to be topologically equivalent to the half BBN can be done in such a way that the binary elements of both the first stage and the last stage of alpha can be labeled strictly increasingly from top to bottom.

The following two techniques can be used to construct new HTs.
Figure 3.7. Scatter Network and Omega Inverse Network
1. Permute the binary elements and its associate links within a stage from the the second stage to the second last stage of the half BBN. For example, see Figure 3.8(a) and (b).

2. Add a fixed input pattern and a fixed output pattern to an LRT. For example, (omega)(rho).

An HT has the property that if it is used as a circulating network (see next chapter), then any permutation can be realized by it in a phase of length 2 using the same control algorithm adopted for transversal networks. Comparing Figure 3.8 with Figure 3.7(a), we observe that the scatter network can realize any permutation by 2 passes if it is preceded by the fixed pattern I. The permutation $P$ realizable by an HT is characterized by the condition that $P(\rho)$ is $\omega^{-1}$ realizable.

In next subsection, although our discussion will be based on stripped networks for convenience and consistency, it is not difficult to see that the results hold even if one looses this condition and lets the first and last stage link patterns be arbitrary.
Figure 3.8. Obtaining an HT from a Half BBN

(to be continued)
(continued)
3.5.2. Bit-Controlled Networks with Single Outlet-Address

A digital controlled or delta network is proposed in [Pate79]. We will generalize its construction; however, we only consider the \( n=2^N \) case. We define a single outlet-address bit-controlled network (SABC) to be an \( n \times m \) BEMIN iff each of its binary element, \( b_{ij} \), can be assigned a sign, \( \text{sign}(b_{ij}) \), to be either positive or negative such that the BEMIN satisfies the following condition.

The outlets are labeled by the binary representations of 0, 1, \( \ldots, n-1 \), from top to bottom when the link pattern \( F_m \) is the identity, as addresses. Any inlet can be connected by a path to any outlet, and the path can be constructed by the following "strip-and-send" distributed control.

Let the inlet hold the bit string of the desired outlet address, \( b_1b_2\ldots b_m \). The path is constructed by sending the bit string through \( m \) intermediate binary elements. If the binary element in stage \( S_1 \) receives the bit string \( b_1b_{i+1}\ldots b_m \) at the upper or lower input terminal, it sets its own state to 0 or 1 according to the exclusive-or of the first bit of the bit string, \( b_1 \), and the sign of the binary element, \( \text{sign}(b_{ij}) \), being 0 (or 1), and then strips bit \( b_1 \) from the string and sends the remaining bit string \( b_{i+1}\ldots b_m \) to the next stage through the upper (or lower, respectively) output terminal. The process repeats until the
desired outlet is reached.

**Theorem 3.23:** The class \( \{SABC\} = \{LT\} \), and \( \{SABC^{-1}\} = \{RT\} \).

**Proof:** From the recursive construction of a left-half \( \text{GBBN} \), any inlet will be led to the outlet with address \( b_1b_2\ldots b_m \) by passing through a \( \text{GBBN}_T \) or \( \text{GBBN}_B \) depending on the bit \( b_i \) being 0 or 1 for each level of the recursion. Thus, an LT is a SABC since topological equivalence does not change the structure of the network. Conversely, any SABC must be an LT because the construction of LTs is the only way to obtain a network such that any two paths from two inlets will be led to the same outlet with the same bit string \( b_1b_2\ldots b_m \). Q.E.D.

**Corollary 3.24:** The class \( \{SABC\} \cup \{SABC^{-1}\} \) is PPR—polynomial and robust.

### 3.6. Beyond Partial Transversal Networks and their Reverse Networks

The recursive construction of PTNs gives some classes of networks with structures which are considered to be "simple" in the context of this chapter. In this section, further classes of networks with simple structures are discussed around a novel class of networks having the property of minimum full access. This class of networks can also be uniquely determined with the aid of
3.6.1. Minimum Full Access Networks

A minimum full access networks (MFA) of size \( n \) is an \( n \times m \) BEMIN where \( m \) is minimum such that any one inlet can be connected to any outlet. We only limit to the \( n=2^N \) case.

A basic minimum full access network (BMFA) is recursive defined as follows.

(Base) A 1-stage BMFA (of size 2) is any primitive 2*1 BEMIN.

(Recursion) An \( m \)-stage, \( m=\log(n) \), BMFA (of size \( n \)) is an \( n \times \log(n) \) BEMIN with link patterns defined by one of the following two alternative recursive construction rules.

Rule 1. An \( m \)-stage BMFA consists of an arbitrary permutation pattern \( F_L \) of degree \( n \), followed by a stage of \( n/2 \) binary elements, followed by an unshuffle pattern of degree \( n \), cascaded by two \((m-1)\)-stage BMFAs, a top one BMFA\(_T\) and a bottom one BMFA\(_B\), and then followed by an arbitrary pattern \( F_R \) (Figure 3.9(a)).

Rule 2. The reverse of the network constructed by Rule 1 (Figure 3.9(b)).

It is obvious that the only way to construct an MFA is to follow the recursive construction above with possibly permuting the
Figure 3.9. The Recursive Construction of the MFA Network
binary elements within a stage of a BMFA. Thus, we have the following.

Theorem 3.25: A network is an MFA iff it is topologically equivalent to a BMFA.

Since an LT or an RT of size $n$ has $\log(n)$ stages, and any inlet can be connected to any outlet, it is an MFA. This also follows from Theorem 3.25. But it is equally clear that the reverse statement is not true. The class of the large number of MFAs has some common properties as stated below.

Corollary 3:26. An MFA network of size $n$ has the following properties.
1. It realizes a set of $2n^2\log(n)/2$ permutations, each corresponding to only one possible setting of the network.
2. There is a single path from any inlet to any outlet.
3. All the paths from a first (last) stage binary element to all the last (first, respectively) stage binary elements form a binary tree.
4. Any two disjoint paths which connect two different pairs of inlets and outlets intersect at most one binary element.
5. For any binary element $b_{ij}$ in the network, all the paths from $b_{ij}$ to all inlets to which it can be connected form a binary tree, the left-tree of $b_{ij}$, and all the paths from $b_{ij}$ to all outlets to which
it can be connected also form a binary tree, the right-tree of $b_{ij}$.

To check if a given network is an MFA, one can check the number of inlets and the number of stages, and property 2 in Corollary 3.26. Since there is a single path from any inlet to any outlet, we can solve the corresponding PPR problem to the class of MFAs as follows.

Step 1. Let $i=1$.
Step 2. If inlet $i$ cannot be connected to the desired outlet $j$, output "no" and stop.
Step 3. Delete the path from $i$ to $j$. Let $i=i+1$. If $i=n$, then output "yes" and stop; else go to step 2. Q.E.D.

Theorem 3.27: The class \{MFA\} is robust and PPR-polynomial.

3.6.2. Rearrangeability Considerations

For any LT alpha, $(alpha)(alpha^*)$ is rearrangeable. For any RT beta, $(beta)(beta^*)$ is rearrangeable. We do not know about other combinations. More restrictedly, let an LT alpha and an RT beta be conjugate. Then $(alpha)(beta)$, or $(beta)(alpha^*)$ is rearrangeable. We do not know whether $(alpha)(alpha) = (alpha)(beta)$, $(beta)(beta) = (alpha^*)(beta)$, or $(beta)(alpha) = (alpha^*)(beta^*)$ is rearrangeable. However, we will give several networks of size 8 which are shown to be rearrangeable by running computer programs to
generate all permutations corresponding to all possible states of all binary elements in the networks. See [Nije75] for some programs available.

Let \( \alpha \) be a stripped omega network of size 8. Since \( \alpha \) is a LRT, \((\alpha)(\beta)\) and \((\beta)(\alpha)\) is rearrangeable if \( \beta = \alpha^{-1} \). By running a program, we show that \((\alpha)(\alpha)\) is also rearrangeable. See Figure 3.10(c). Hence obviously, \((\omega)(\text{stripped } \omega)\), Figure 3.10(b), and an omega cascaded by another omega, Figure 3.10(a) are rearrangeable. It also implies that \((\beta)(\beta)\) or \((\text{stripped } \omega)^{-1}(\text{stripped } \omega)^{-1}\) is rearrangeable. It is unknown whether \((\text{stripped } \omega)(\text{stripped } \omega)\) of any size can realize all permutations. Table 3.2 gives the number of permutations realized by the partial \((\text{stripped } \omega)(\text{stripped } \omega)\) networks of size 8.

Let \( \alpha \) be an LT of size 8 which is not an RT as shown to be the left 3 stages in Figure 3.11(a). Let \( \beta \) be the reverse network of \( \alpha \) as shown to be the right 3 stages in Figure 3.11(a). Then Figure 3.11(a) is a transversal network and \((\alpha)(\beta)\) is rearrangeable. Probably surprisingly, computer programs show that \((\alpha)(\alpha)\), \((\beta)(\beta)\), and \((\beta)(\alpha)\) are all rearrangeable although none of them is a transversal network. See Figure 3.11(b),(c),(d). More generally, is it true
Figure 3.10. Small Rearrangeable Networks based on LRTs
Table 3.2. Permutation Number of Partial \((\Omega)^2-1\) of Size 8

<table>
<thead>
<tr>
<th># of stages</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td># of permutations</td>
<td>16</td>
<td>256</td>
<td>4096</td>
<td>18688</td>
<td>40320</td>
</tr>
</tbody>
</table>
that any minimum full access network of any size cascaded by another minimum full access network of the same size, possibly merging the middle two stages, always form a rearrangeable network? It seems very difficult to pursue this.

3.6.3. Adding an Arbitrary Stage to the Minimum Full Access Network

Let \( \alpha \) be any minimum full access network of size \( n \). An \( \text{MFA}+1 \) network of size \( n \) is constructed by cascading a one-stage \( \text{BEMIN} \) of arbitrary link pattern \( F_0 \) and \( F_1 \) of size \( n \) to \( \alpha \).

**Theorem 3.28:** The class of all MFA+1 networks is PPR-polynomial and robust.

**Proof:** Consider an instance of the corresponding PPR problem having an MFA+1 and a permutation \( P \). Let the outlets be labeled according to \( P \). It is clear that any inlet has exactly 2 ways to connect to its desired outlets. Thus, a polynomial algorithm similar to what solves 2-SAT problem or 2-Color problem can solve the PPR problem. We will use the algorithm which solves the timetable problem with binary teachers [Even76] here since it can be parallelized effectively.

The algorithm will progressively connect an inlet to the desired outlet until either all inlets have been connected or it is
Figure 3.11. Small Rearrangeable Networks based on LTs & RTs
(continued)

(beta)(beta)

(c)

(beta)(alpha)

(d)
found that the connection for all inlets simultaneously is impossible.

Step 1. Set Flag=2.

Step 2. If all inlets have been connected to their desired outlets, then output "yes" and stop.

Step 3. If there is an inlet for which there is no path to connect it to its desired outlet, go to step 7.

Step 4. If there is no inlet for which there is only one possible path to connect it to its desired outlet, go to step 6.

Step 5. Let $I_1$ be an inlet which has only one way to connect to its desired outlet. Temporarily connect $I_1$ and go to step 2.

Step 6. Make all temporary connections permanent. Let $I_1$ be an inlet which has not been connected. Arbitrarily choose a path to connect $I_1$ to its desired outlet and record this decision. Set Flag=1 and go to step 2.

Step 7. If Flag=2, output "no" and stop.

Step 8. Undo all temporary connections and use another path for the recorded inlet to connect it to the desired outlet. Set Flag=2 and go to step 3.

Note that new connections are set by entering step 6 and this occurs only when no inlet has a single way to connect to its desired outlet. Since there are at most two ways for an inlet to be connected to its desired outlet, the algorithm is sufficient. Q.E.D.
Corollary 3.29: The class \( \{\text{PTN}\} \cup \{\text{PTN}^{-1}\} \cup \{\text{MFA}\} \cup \{\text{MFA}+1\} \) is robust and PPR-polynomial.

3.7. Summary

We define a class of binary element multistage interconnection networks to be PPR-polynomial or efficiently permutation-recognizable if given a network in the class an algorithm can determine whether it can realize a given permutation in polynomial time. The class is said to be robust or efficiently membership-recognizable if a polynomial time algorithm can decide if a given network is in the class. Along with the searching of efficient membership-recognition and permutation-recognition classes of BEMINs, we define many novel as well as some generalized classes of networks with good properties in this chapter.

Starting from Hall’s theorem and applying the property of topological equivalence and bunch connectivity, and a canonical labeling, we find that the class of partial transversal networks and their reverse networks is both robust and PPR-polynomial. Important characterizations including the number of permutations and the transformation of permutations between networks of PTN and PTN\(^{-1}\) are obtained. The existence of conjugate networks allows us to relate some subclasses of PTN and PTN\(^{-1}\) to some subclasses of minimum full access networks, and the class of single outlet-address
bit-controlled networks and their reverse networks. The results previously presented in the literature for some particular classes of networks are thus unified and generalized and some specific new networks are figured out.

Attempting to move beyond PTN and PTN\(^{-1}\), some small networks which are not PTN or PTN\(^{-1}\) but are rearrangeable are presented. We further find that the class of MFA and the class of networks obtained by adding one stage to MFA are also robust and PPR-polynomial. We also discuss the properties of these classes of networks.
Chapter 4

THE CAPABILITY AND EFFICIENCY OF CIRCULATING INTERCONNECTION NETWORKS

In the previous chapters, we have shown that it is difficult to come up with an efficient systematic way to check for rearrangeability if an $n \times m$ BEMIN is given as input. We have not considered the more restricted problem of inputing BEMINs with the same interconnection link pattern between any two adjacent stages. Since a single link pattern is used, one can just build a single stage interconnection network and allow data to circulate through it a number of times to reach their destinations. The minimum stage for rearrangeability problem then becomes the problem of finding the minimum upper bound on the number of circulations (passes) needed to realize any permutation. In connection with this problem, we will consider some generalized and some restricted problems in this chapter.
4.1. The Definition of Circulating Networks

An interconnection network (ICN) is defined in Chapter 1 as a physical entity which facilitating connection and communication between the terminals in a specified set of terminals \( \{T_1, T_2, \ldots, T_n\} \). We call the connection and communication status of the set of terminals of an ICN at a particular time its state (function) at that time. A state can be specified by \( \{A_0, A_1, \ldots, A_p\} \) where \( \{A_i: 0 \leq i \leq p\} \) forms a partition of the set of terminals, and \( A_0 \) represents the set of idle terminals and for each \( A_i = \{T_{i0}, T_{i1}, \ldots, T_{iq}\}, 1 \leq i \leq p \), a connection is assumed, with \( T_{i0} \) representing the sender and \( T_{ij}, 1 \leq j \leq q \), receivers. The capability of an ICN depends on its possible states. As defined in Chapter 1, a partitioning network is an ICN with full capability. For economic reasons, many interconnection networks in a specific environment are designed with limited capability and sometimes a set of unit messages (data) must be routed to the desired destinations by means of experiencing a sequence of states of the network.

We define an \( n \)-by-\( n \) one-way two-sided pairwise-connected ICN capable of performing a set \( S = \{s_1, s_2, \ldots, s_m\} \) of states without idle terminals as a **bijection state network** (BSN). Suppose that we denote the set of inlets as \( I = \{I_1, \ldots, I_n\} \) and the set of outlets as \( O = \{O_1, \ldots, O_n\} \), then each state defines a permutation (bijection)
which maps I to O. Each state can be controlled by a control line, or the control lines can be encoded such that k control lines can control \(2^k\) states. The view of a BSN from its state set S allows us to abstract the BSN such that the actual implementation is not seen for the time being.

The BSN is used in the circulating network (CN) shown in Figure 4.1. The set of input data at \(I_{N_1}, I_{N_2}, \ldots, I_{N_n}\) is passed once through the multiplexor to \(I_1, I_2, \ldots, I_n\), and then through the BSN, set at a state defined by a bijection, to \(0_1, 0_2, \ldots, 0_n\). If the desired permutation of data has not been achieved, the demultiplexor and the multiplexor will be selected for "up" path, and the data will be routed through the BSN again to \(0_1, 0_2, \ldots, 0_n\). The state of the BSN may be changed for each circulation. Until the desired permutation is achieved, the demultiplexor and the multiplexor will be selected for "down" path, and the data will be sent to \(O_{OUT_1}, O_{OUT_2}, \ldots, O_{OUT_n}\) and the next set of data can optionally comes in. Note that conceptually there is no storage element in the CN by definition. In practice, there may exist delays or pass transistors [Mead80] for synchronization to assure that the set of data flow together and the state setting of the BSN is always achieved before the set of data reaches \(I_1, I_2, \ldots, I_n\) and is retained so until the set of data is routed to \(0_1, 0_2, \ldots, 0_n\). Thus the CN is completely defined by its BSN, and the CN is implementation-independent in the
sense that the BSN is given by the set of realizable states.

The set of the states of a BSN is defined as the configuration of the CN which it is in. Consider a set of input data at the inlet side of a CN to be sent to the outlets according to a permutation P. It is understood that the BSN may realize the permutation P by a number of passes (circulations) of the set of data. When the set of outlets receives the set of data sent by the inlets and the desired permutation P of the input data is achieved at the outlets, a phase of the CN is completed. A phase contains a sequence of states and for each state the set of data is routed through the BSN one time. The length of the state sequence of a phase is called the phase length. We call a CN capable of performing possible phases for all permutations a (full) permutation circulating network (PCN).

4.2. Variations of Circulating Networks and Examples

The CN can be enhanced by adding a mask register \( M = [M_1, M_2, \ldots, M_n] \) at \( I_1, I_2, \ldots, I_n \) between the multiplexor and the BSN. If a particular inlet \( I_i \) is masked (disabled), \( M_i = 0 \), for a circulation, the data at \( I_i \) will be prohibited from sending into the BSN, and the content of data at \( I_i \) will be updated if there is a data recirculating from the outlet \( O_i \). Such a CN will be called a masked CN. We will assume that the mask register can disable any arbitrary subset of the set of inlets. In some situation, it may be
Figure 4.1. Circulating Network
economical to have limited masking functions, such as the address mask discussed in [Sieg77a] for several specific types of interconnection networks.

We generalize the BSN to be a **mapping state network** (MSN), if each state of the MSN is capable of connecting an assignment (a mapping) of all outlets to some inlets. That is, the set of all possible states, not necessarily realizable by a particular MSN, contains all \( n^n \) mappings of outlets to inlets. An **extended circulating network** (ECN) can be constructed in the same way as a circulating network shown in Figure 4.1 except that the BSN is replaced by an MSN, and a mask register is added to block some data of some inlets to be sent through the MSN in case the MSN is controlled in a state which realizes a multicasting (replicating) function. Similarly, an ECN which has possible phases corresponding to all \( n^n \) assignments will be called a **generalized extended circulating networks** (GECN).

Storage elements can be added into CN or ECN to make it become a "buffered" CN or ECN. We define a **buffered CN** as a CN with a buffer (swap) register \( B = [B_1, B_2, \ldots, B_n] \) added at \( O_1, O_2, \ldots, O_n \) between the BSN and the demultiplexor. The swap register augments the capability of the CN by following the rule below. Before each circulation of data, the data at \( O_1, 1 \leq i \leq n \), can be stored into or
swapped with the content in \( B_1 \) of the swap register. The case of ECN is similar. Figure 4.2 shows a buffered ECN.

A common bus can be viewed as an ECN where the MSN consists of all \( n \) broadcasting states and each state allows one inlets to send data to all outlets. A two-way ring or loop is an ECN which contains two cyclic shift states, some neighbors exchange states and some multicasting states in which one inlets can send data to its +1 and −1 outlets simultaneously. The common bus and the ring are inefficient for permuting data.

The mesh-connected network for ILLIAC IV [Kuck68] is a CN with configuration \( \{s_n, s_w, s_e, s_s\} \), where each state permutes data to follow the north, west, east, or south direction such that the inlet \( i \) sends data to the outlet \( i-\text{SQRT}(64), i-1, i+1, \) or \( i+\text{SQRT}(64) \), arithmetic modulo 64, \( 0 < i < 63 \). The "single-stage" networks or recirculating networks defined in [Sieg79] including PM2I, shuffle-exchange, and cube types are some examples of bijection state networks which can be used in a circulating network when their states are clearly defined.

Multistage interconnection networks can also be used as the BSN in a CN. By using the reverse-exchange network [Feng79], the CN can

\[ \text{SQRT}(x) \] denotes the square root of \( x \).
Figure 4.2. Buffered Extended Circulating Network
realize any permutation with phase length at most 2. As indicated in the previous chapter, the network obtained by cascading one reverse-exchange network to another is a transversal network which is rearrangeable. The same technique can be applied to the omega-rho network which is defined as an omega network [Lawr75] followed by a bit-reversal (rho) link pattern. The above concepts and other concepts are unified and generalized in Chapter 3. If one uses a permutation network as the MSN of a ECN, the ECN can be used as a partitioning network, since there is a cyclic permutation corresponding to every partition of a set.

4.3. Problems Related to the Design of the Circulating Network (CN)

In the systematic way of designing a circulating network, we face a number of problems. The number of inlets and outlets may vary. The cost of a CN depends on the size of its configuration and whether the states of the configuration can be implemented effectively. The number of permutations and the types of permutations a CN can realize define its functional capability. The efficiency of a CN depends on if it has a short sequence of states to realize a desired permutation. The reliability can be interpreted as the capacity a CN can still achieve its specified capability within some predefined specification of efficiency even if some states are erroneous. The similar problems occur for the
In this chapter, our focus is mainly on capability and efficiency. The following theorem shows that it only requires two states to achieve a PCN. But it may require many states to realize a desired permutation.

Lemma 4.1: A CN which has configuration \{(1,2), (2,3), \ldots, (n-1, n)\} or \{(1,2), (1,2,\ldots,n)\}, where each state is denoted by the cyclic permutation notation, is a PCN.

Proof: Every permutation can be expressed as a product of transpositions [Hall59], where a transposition is a 2-cycle, i.e. a permutation which just interchanges (exchanges) two points. For \(1 \leq j < k < n\), we have \((j, k+1) = (k, k+1)(j, k)(k, k+1)\), so that \((j, k+1)\) can be obtained from \((j, k)\) with the aid of the transposition \((k, k+1)\) of successive points. This shows that we do not need every transposition, and the symmetric group \(S_n\) can be generated by the transpositions \((k, k+1)\) of successive points, \(1 \leq k \leq n\).

For \(0 \leq r \leq n-2\), we have \((1,\ldots,n)^r(1,2)(1,\ldots,n)^{-r} = (r+1, r+2)\). Hence, the two states \((1,2)\), and \((1,2,\ldots,n)\) can generate every permutation. Q.E.D.

We will discuss the following problems for circulating networks. The problems associated with the ECN can be similarly
defined.

1. The CN capability problem: Given a CN configuration $\{s_1, s_2, \ldots, s_m\}$, determine if it can realize a given permutation $P$.

2. The PCN determination problem: Given a CN configuration $\{s_1, s_2, \ldots, s_m\}$, determine if the CN is a PCN.

3. The shortest phase length (SPL) problem: Given a CN configuration $\{s_1, s_2, \ldots, s_m\}$ and a permutation $P$, find the shortest phase length for the CN to realize $P$. (If $P$ is not realizable, the phase length is defined to be infinity.)

4. The CN efficiency problem: Given a CN configuration $\{s_1, s_2, \ldots, s_m\}$, find the minimum upper bound on the phase length needed to realize any permutation which can be realized by the CN.

In the following two subsections, we will firstly show that the first two problems are polynomially solvable, and then that the last two problems are NP-hard problems [Gare79]. As defined in Chapter 2, problems polynomially solvable are easy problems, and NP-hard or NP-complete problems are deemed to be hard.
4.3.1. The CN Capability Problem and the Permutation Circulating Network (PCN) Determination Problem

The CN capability problem is solvable in polynomial time because checking whether a given permutation is a member of the permutation group generated by a given set of permutations is easy.

Theorem 4.2: The CN capability problem is easy.

Proof: The algorithm is given in [Sims70] and shown to be polynomial time in [Furs80].

Let $G$ be the group of permutations realizable by the CN of size $n$ with configuration $\{s_1, \ldots, s_m\}$. $G$ can be written as $G = (G_0/G_1)(G_1/G_2) \cdots (G_{n-1}/G_n)$, where $G = G_0$ and $G_i$ is the subgroup of $G$ fixing $1, \ldots, i$. The algorithm will firstly use $s_1, \ldots, s_m$ as generators to obtain all the coset representatives in canonical forms for $G_i/G_{i+1}$, $i=0,1,\ldots,n-1$. This consists of two steps. The first step "sifts" the generators and the second step sifts the products of all existing pairs. Sift is a program which takes the input permutation into a coset representative by checking with existing coset representatives and add it to the set of coset representatives if it is new.

The following procedure uses the upper triangle of an $n$ by $n$ table. Initially, each entry $\text{TABLE}[i,j]$, $1 \leq i \leq n$, contains the
identity permutation and all other entries are empty. A permutation $P=(p_1, \ldots, p_n)$.

Procedure SIFT(X);

```plaintext
var
    i: integer;
    X: permutation;
    TABLE: array [1..n; 1..n] of permutations;
begin
    i:=0;
    while {i#n-1 and
        table[i,j]_i+1=x_i+1, j=i, i+1, ..., n} do
        begin
            i:=i+1;
            X:=TABLE[i, j]-1X;
        end;
    if {X#TABLE[i,j], j=i, i+1, ..., n} then
        if {TABLE[i,k], k=i, i+1, ..., n, empty} then
            TABLE[i, k]:=X;
end;
```

Determining if the permutation is realizable by the CN is equivalent to checking if $P$ is in $G$. Since the coset representatives have been obtained in their canonical forms, one can just sift $P$ to see if it can be written in canonical form without introducing a new coset representative.

Q.E.D.

Theorem 4.3: The PCN determination problem is easy.

Proof: The same algorithm in the above theorem is used to obtain all the coset representatives. The coset representatives in $G_i/G_{i+1}$ only differ by where they map the letter $i+1$ with $1, \ldots, i$ fixed. The order of the group $G$ generated is the product of the sizes of $G_i/G_{i+1}$.
The CN is a PCN iff the set of the realizable permutations is the symmetry group $S_n$ of order $n!$. Thus, the CN is a PCN if and only if the number of coset representatives in $G_i/G_{i+1}$ is $n-i$. Q.E.D.

4.3.2. The Shortest Phase Length Problem and the CN Efficiency Problem

We define the decision problem of SPL as follows. Given a CN configuration $\{s_1, s_2, \ldots, s_m\}$, a permutation $P$, and an integer $K$, determine if the CN has a phase of length $h$ realizing $P$ such that $h < K$.

Theorem 4.4: The decision problem of SPL is NP-complete even if the CN in the SPL is a PCN.

Proof: We show the NP-completeness for the special case. It is clear that SPL is in NP. It has been shown [Karp72] that EXACT COVER BY 3-SETS (XC3) is NP-complete, and we will reduce XC3 to SPL and use the notations in [Gare79].

An instance of XC3 consists of a set $X$ with $|X| = 3q$ and a collection $C$ of 3-element subsets of $X$. The XC3 problem is to determine whether $C$ contains an exact cover for $X$, i.e., a subcollection $C'$ contained in $C$ such that every element of $X$ occurs in exactly one member of $C'$.
Let $X = \{x_1, \ldots, x_{3q}\}$ and $C = \{C_1, \ldots, C_r\}$ be an instance of XC3. We construct the following instance of SPL.

Let the number of inlets of the CN be $6q$ and partition the set of inlets into $3q$ subsets such that the subset $\{2i-1, 2i\}$ corresponds to $x_i$. For every $C_j$, $1 \leq j \leq r$, we construct a state $s_j$ of CN such that $s_j$ exchange $2i-1$ and $2i$ iff $x_i$ is in $C_j$. Let the configuration of the CN be $\{s_1, \ldots, s_r, s_{r+1}, s_{r+2}\}$, where $s_{r+1} = (1, 2)$ and $s_{r+2} = (1, 2, \ldots, 6q)$ in cyclic permutation notation. Clearly, the CN is a PCN from Lemma 4.1. Let the permutation $P$ be $(2, 1, 4, 3, \ldots, 6q, 6q-1)$, that is, $P$ permutes all pairs in all subsets $\{2i-1, 2i\}$, $1 \leq i \leq 3q$. Let $K = q$.

(a) Assume that the XC3 instance has an exact cover $C' = \{C_{j_1}, C_{j_2}, \ldots, C_{j_q}\}$ of $q$ subsets. The permutation $P$ is realizable by the phase of state sequence $s_{j_1}, s_{j_2}, \ldots, s_{j_q}$, and the phase length is $K$. Thus, the answer to the SPL instance is "yes".

(b) If the CN constructed has a phase $F$ which realizes $P$ with phase length less than or equal to $K$ ($=q$), then since there are $3q$ pairs to be exchanged, $F$ cannot include the state $s_{r+1}$ or $s_{r+2}$, or both. Furthermore, the phase length must be $K$ which equals $q$, since each $s_i$, $1 \leq i \leq r$, can only exchange exactly 3 pairs. Let $F = s_{j_1}, s_{j_2}, \ldots, s_{j_q}$, then the corresponding $C' = \{C_{j_1}, C_{j_2}, \ldots, C_{j_q}\}$ is an exact cover for $X$. Q.E.D.
From the above Theorem, the generalized SPL problem for an ECN and a given assignment of outlets to inlets is also NP-complete.

**Theorem 4.5:** The CN efficiency problem is NP-hard.

**Proof:** It has been shown in [Even81] that the following problem is NP-hard.

Minimum upper bound on generator sequences (MBGS): Given a set of generators of a permutation group G, find the minimum upper bound on the -length of generator sequences needed to realize any permutation in G.

Hence, we have the theorem by a direct substitution where the set of generators is replaced by the CN configuration, and the length of generator sequences is replaced by the phase length. Q.E.D.

4.4. Using a Binary Element Multistage Interconnection Network as the Bijection State Network in a CN

The specification of the problems in previous sections is made by the configuration of the circulating network. When a binary element multistage interconnection network (BEMIN) (see the definition in Chapter 1) is used as the BSN in a CN, the CN can (or should) be specified by the link patterns of the BEMIN. If this is the case, new problems are posed. In this section, we study
single-stage BEMIN first, and then the multistage cases. We distinguish strongly-connected networks from full access networks, and rearrangeable networks from strict-sense nonblocking networks. We solve the PCN determination problem and the CN capability problem when the CN is specified by the link patterns of the BEMIN.

4.4.1. Single-Stage Circulating Networks

An $n \times m$ BEMIN is defined in Chapter 1. A single-stage BEMIN is an $n \times 1$ BEMIN specified by $F_0 S_1 F_1$. In a circulating network application, it is reasonable to assume that $F_0$ or $F_1$ is the identity. Thus, our focus is the single-stage BEMIN as shown in Figure 4.3(a) or Figure 4.3(b), where $x = F_1 F_0$. It will be seen that all these three schemes are equivalent in many design problems. Such a CN is completely determined by the given link pattern $x$, and it is thus called "implementation-dependent", which means a CN is only built by using a single-stage BEMIN. We also use the term "single-stage CN" for short. We say "the stage" when we mean the stage itself with both $F_0$ and $F_1$ being the identity. The stage is of size $n$ ($n$ even) if it has $n$ inlets and $n$ outlets, and a column of $n/2$ binary elements or nodes (see Chapter 2).

If permutations $x_1, \ldots, x_r$ generate a group $B(n, r)$ with relations $g^n = 1$ (the identity) for every $g$ in $B(n, r)$, then this group is called the Burnside group of order $n$ with $r$ generators [Ha1159].
Figure 4.3. The BSN of Single-Stage Circulating Networks
Lemma 4.6: The set of permutations realized by the stage of size n forms a Burnside group of order 2 with n/2 generators.

Proof: There are two obvious way to obtain the generators. One, permutations (1,2),(3,4),..., and (n-1,n) in the cyclic permutation notation, and the other, (1,2),(1,2)(3,4),..., and (1,2)(3,4)...(n-1,n). It is clear that every element generated besides the identity is of order 2. Q.E.D.

Lemma 4.7: The set of permutations realized by the stage of size n forms an Abelian group of order 2n/2.

Proof: The Burnside group of order 2 is Abelian, because xy=(xy)^-1=(y)^-1(x)^-1=yx. The n/2 generators are all independent and form a basis. Q.E.D.

Theorem 4.8: Any single-stage BEMIN of size n has exactly 2n/2 distinct states.

Proof: Let B be the group of the stage. Then B is a subgroup of the symmetry group S_n. Since the link pattern x is in S_n and it is fixed, Bx corresponding to Figure 4.3(a) is a left coset of B, and xB corresponding to Figure 4.3(b) is a right coset of B, with x being the representative of the coset. It follows from the fact that the left (right) coset of a subgroup H has the same cardinal number of elements as H. Q.E.D.
Corollary 4.9: The CN with a single-stage BEMIN has a configuration $xB$ or $Bx$ of $2n/2$ states.

Since $(Hx)^{-1} = x^{-1}H$, there is a one-to-one correspondence between the left coset and the right cosets of $H$. The correspondence in our network terminology is between a network and its reverse network. It is known that two left (right) cosets are either disjoint or identical sets of elements. Since the configuration of a CN completely determines the CN, we have the following from Lagrange theorem.

Corollary 4.10: There are $(n-1)(n-3)...(3)(1)(n/2)(n/2-1)...(1)$ different CNs of size $n$ using the single-stage BEMIN $Bx$ (or $xB$).

Among all these CNs, which of them are PCNs? One can get the CN configuration and use the algorithm for PCN determination problem to solve this question. However, since a single-stage CN is also completely determined by the link pattern $x$, it can simply be specified by the link pattern $x$. Because it takes exponential $O(2n/2)$ time to get all the states of the CN, we have a new problem. (As a matter of fact, the four problems discussed in the last two sections pose new problems in this implementation-dependent case.) In this case, there are $n!$ different single-stage CNs. What patterns can realize all permutations by a finite number of passes? That is, if a single-stage CN is specified by its pattern $x$, we try
to decide whether it is a PCN. We will give good characterizations and good (polynomial time) algorithms to this problem. A good characterization is a property P which we are able to prove P efficiently if it holds and to disprove it efficiently if it does not [Lova79].

4.4.2. Strongly-Connected Networks

A one-way two-sided interconnection network is strongly connected (dynamic full access) iff any inlet can be connected to any outlet by means of a finite number of passes through the network. This dynamic full access property is used as a criterion for fault tolerance analysis in [Shen82] for some specific types of interconnection networks composed of binary elements. In our case of BEMIN as BSN, it is clear that a network is strongly connected if and only if its corresponding BEMIN cyclic digraph (see Chapter 1) for realizing the identity permutation is strongly connected.

Theorem 4.11: Any buffered CN is a PCN iff its BSN is strongly-connected.

Proof: The necessity is obvious. For sufficiency, a straight-forward way to realize a permutation is the following. Firstly, store all data into the buffer register. Then, choose a data in B^ which is not in the correct position, swap it out to 0^,
and send it to the correct outlet $O_j$. Swap $B_j$ and $O_j$ and send it to the correct outlet $O_k$; if $k$ is not equal to $i$, then let $j=k$ and repeat this step until $i=k$. The above step corresponds to a cycle in the permutation. Choose a data which is not in the correct place to start another cycle. Repeat until all data are in the correct position. Send the buffer register to the outputs $OUT_1, \ldots, OUT_n$.

Q.E.D.

Since it only takes polynomial time to decide if a digraph is strongly-connected [Aho74], we have the following.

**Corollary 4.12**: It is easy to decide if a buffered CN constitutes a PCN.

The simplest buffered PCN can have a configuration with just one state which is a permutation of cycle length $n$. It is topologically equivalent to a static ring (loop) as shown in Figure 4.4 for size 8, where the nodes denote both the inlet $I_i$ and outlet $O_i$ of the BSN, the squares represent the buffer register, and the arrows indicate the mapping from $I_i$ to $O_j$, $1 \leq i \leq n$, of the points of the permutation. The loop itself functions as a circular shift register.

However, the strongly-connected property is not a sufficient condition for a CN to be a PCN. Figure 4.5 shows some single-stage
Figure 4.4. Ring Topology
BSNs which are strongly-connected, but they do not constitute a PCN when used in a CN. It is not difficult to check that some permutations are not realizable by a finite number of passes through a BSN in the figure. The first one, (a), is a "double-loop", and the others are "cross-overs".

Also note that the link pattern of Figure 4.5(a) is a permutation similar or conjugate [Hall59] to a cyclic-shift permutation in Lemma 4.1. While the cyclic-shift pattern constitutes a PCN, the double-loop pattern does not.

4.4.3. Full Access Networks

An $n \times m$ BEMIN has the property of full access or full connectivity iff there is a path to connect any inlet to any outlet. It is immediately seen that the condition is equivalent to that any binary element in stage $S_1$ has a path to any binary element in stage $S_m$. Apparently, if an $n \times m$ BEMIN alpha has full access then the BEMIN built by adding more stages to alpha is still a full access network.

Consider an $n \times m$ BEMIN alpha used as the BSN in a CN. If a permutation is realized by the CN with phase length $p$, it is equivalent that the $n \times (mp)$ BEMIN beta constructed by cascading $p$ alpha's, a repeated pattern network (RPN), can realize the same
Figure 4.5. Some Non-PCN Strongly-Connected Networks
permutation. That is, alpha can simulate beta by a number of passes. A single-stage CN simulates an n*m BEMIN, with link patterns \( F_1, \ldots, F_m \) all equal to the pattern of the BSN and \( F_0 \) being the identity, by phase length \( m \).

A circulating network has **full access** iff there is a repeated pattern network being simulated having full access. An inlet \( I_1 \) of a CN has **full access** iff there is a repeated pattern network being simulated has the property that the inlet \( I_1 \) can be connected to any outlet. One can optionally take the simulated RPN with minimum number of stages in both definitions. Thus, the minimum phase length to achieve full access of a CN is the largest value taken from the minimum phase lengths achieving full access of all inlets of the CN.

**Theorem 4.13:** A single-stage CN has full access iff there is an inlet having full access.

**Proof:** The necessary condition is obvious. Consider a single-stage network \( Bx \) of size \( n \). Let the binary element be denoted by \( a_1, a_2, \ldots, a_{n/2} \) from top to bottom. We add a column of binary elements \( b_1, b_2, \ldots, b_{n/2} \) at the right-hand side of the link pattern \( x \) to aid us to imagine the circulation of the network. Suppose that \( a_1 \) has full access. Let \( R \) be the set of nodes having full access. Thus, \( R \) includes \( \{a_1\} \). One of the links connected to \( a_1 \) must be led
to a $b_j$, $j \neq i$. If not, no full access is possible for $a_i$. Now, consider $b_i$, which is equivalent to $a_i$ at the same level. At least one link must be led to $b_i$ from a node $a_k$, $k \neq i$, since at least one link from $\{a_i\}$ must be led to a node not in $\{a_i\}$. Thus, $\{a_i, a_k\}$ is contained in $R$.

Suppose $R_m = \{a_{i_1}, \ldots, a_{i_m}\}$ including $a_i$ is contained in $R$, $m \leq n$. There must be a link from $R_m$ to a node $b_j$, $j \neq i_1, \ldots, i_m$. Thus, there exists a link from a node $a_k$, $k \neq i_1, \ldots, i_m$, to one of the nodes in $R_m$. We have a full access set $R_{m+1} = \{a_{i_1}, \ldots, a_{i_m}\} \cup \{a_k\}$. By induction, $R$ is the set of all nodes, and the sufficient condition is proved.

Q.E.D.

If we let each binary element have two additional states as shown in Figure 1.10. Then, the above theorem can be interpreted as that if a inlet has broadcast function then all inlets have broadcast function in an ECN.

**Theorem 4.14:** A single-stage CN is a PCN iff it has full access.

**Proof:** The necessity is obvious. Let $B_x$ be the single-stage network of size $n$. Let $\alpha$ be a multistage $n \times m$ BEMIN which is simulated by $B_x$ and has full access. The proof is based on the fact that any permutation can be expressed as a product of a sequence of
transpositions. We will show that any transposition \((i,j)\) is realizable by the circulating network.

Let inlet \(i\) be routed to an outlet \(p\) connected to a binary element \(b_{pm}\) in alpha. This can be done by a path since the \(n\times m\) BEMIN has full access. Now, try to route the inlet \(j\) to another outlet \(p'\) connected to the same binary element \(b_{pm}\). There are two cases.

Case 1. There is a disjoint path from \(j\) to \(p'\).
Case 2. There is no disjoint path from \(j\) to \(p'\) and the path conflicts with the path from \(i\) to \(p\) at a binary element \(b_{qk}\) in the \(k\)th stage, \(k<m\).

In case 1, route other inlets arbitrary to outlets. The \(n\times m\) BEMIN now realizes a permutation \(P\). It is known that the identity permutation will be realized by \(c\) passes through alpha of the same setting, where \(c\) is the least common multiple of the lengths of the cycles of \(P\). Thus, the transposition \((i,j)\) is achieved by complementing the state of \(b_{pm}\) for only the first pass.

Suppose that case 2 occurs. Take the first \(k\) stages of alpha to form an \(n\times k\) BEMIN beta. Route the inlet \(i\) to outlet \(q\) and route the inlet \(j\) to \(q'\); \(q\) and \(q'\) are connected to \(b_{qk}\). Route other inlets arbitrary to outlets of beta. The setting of beta realizes a permutation \(Q\). The same technique in case 1 can be used here for
beta to realize the transposition \((i,j)\). Hence, the sufficient condition is satisfied. Q.E.D.

4.4.4. The PCN Determination Problem for Single-Stage CNs

Once the PCN condition is characterized for single-stage CN, we pursue the problem of efficient checking for the condition.

Theorem 4.15: Given the link pattern of a single-stage CN, it is easy to determine if the CN is a PCN.

Proof: Let the binary elements be labeled \(1,2,\ldots,b\) from top to bottom, where \(b=n/2\).

Step 1. Let the set \(R_0=\{i\}\), where \(i\) is any integer between 1 and \(b\). \((R_0\) is the set of starting base nodes.\)

Step 2. Let the set \(R_1=R_0\). \((R_1\) is the temporary set of accessible nodes.\)

Step 3. Find all the accessible nodes from \(R_1\) by a circulation and denote them by the set \(R_2\).

Step 4. If \(|R_2|=b\), then output "yes" and stop.

Step 5. If \(R_2=R_0\), then output "no" and stop.

Step 6. If \(|R_2|=|R_1|\), then let \(R_1=R_2\) and go to step 3.

Step 7. \((|R_2|>|R_1|)\) Let \(R_0=R_2\) and go to step 2.

The algorithm is base on Theorem 4.13 and 4.14. It tests an arbitrary inlet for full access. It is clear that when the
algorithm answers "yes" at step 4, all the nodes are accessed by the
chosen node x. The algorithm answers "no" only when the condition of
step 4 is false, and when the set of base nodes is resumed; thereby
the further retrace will produce the same situation and it can never
reach all nodes.

The loop between step 2 and step 7 will be executed at most b
times, since the number of reachable nodes always increase when the
execution reaches step 7. A new base set $R_0$ is formed before the
algorithm enters the loop between step 3 and step 6. The loop
repeats only when no increase of the number of reachable nodes and
when the set of base nodes is not resumed. Let $R_1$ access $R_2$ at an
iteration of the loop. Then $|R_2| \neq b$, $R_2 \neq R_0$, and $|R_2| = |R_1|$ iff the
loop will repeat. Thus, $R_2 \neq R_1$, if the loop repeats, because there
is at least one node in $R_1$ led by a link from a node in the previous
$R_1$. Suppose the loop will repeat. For any node $x_1$ in $R_1$ not in $R_2$,
$x_1$ will never be included in the set $R_1$ or $R_2$ by further iterations
of the loop. If not, since $x_1$ is led from a node not in $R_1$, either
$|R_2| > |R_1|$ by the pigeonhole principle on the links out from $R_1$ to $R_2$
or $R_2 = R_1$ in the next iteration. This is a contradiction. Thus,
since there is at least one $x_1$ in $R_1$ not in $R_2$ if the loop repeats,
the loop between step 3 and step 6 will be executed at most b times.

Q.E.D.
As an example, the trace of the algorithm for the network of Figure 4.6 is shown by the starting set $R_0 = \{1\}$, then $R_1 = \{1\}, \{2,3\}, \{4,5\}, \{6,7\}, \{1,2,3\}, \{2,3,4,5\}, \{4,5,6,7\}, \{1,2,3,6,7\}, \{1,2,3,4,5\}, \{2,3,4,5,6,7\}$, and finally $R_2 = \{1,2,3,4,5,6,7\}$. Thus, the CN uses Figure 4.6 as the BSN is a PCN. As another example, any single-stage CN of size $n$ with a link pattern which is a $2n/2$-cycle (see Chapter 1) is a PCN.

4.4.5. The CN Capability Problem for Single-Stage CNs

Given a single-stage CN, is it difficult to determine whether it can realize a given permutation $P$, if the single-stage CN is specified by the link pattern of its BSN?

From Corollary 4.9, the set of the permutation a single-stage CN of size $n$ is the permutation group generated by the $2n/2$ states in the configuration of the CN. Let the configuration of the CN be $Bx$. The generated group, $\langle Bx \rangle$, is completely determined by the link pattern $x$ and we call it a single-stage circulating network group.

The CN capability problem then becomes the problem of determining if a given permutation $P$ is in a given $\langle Bx \rangle$, which is specified by the link pattern permutation $x$.

It is more convenient to consider the full access property of a binary element instead of that of an inlet. From Theorem 4.13, the
Figure 4.6. A BSN for a Permutation Circulating Network
full access property of a binary element implies the full access property of other binary elements. The algorithm given in the proof of Theorem 4.15 checks if an arbitrary binary element has full access.

If the algorithm stops at step 5, the binary element does not have full access and this is done by observing the equality of the starting node set and the newly generated node set. However, although the node does not have full access, it may access a number of nodes, i.e., the set $R_2$ or the starting set $R_0$.

If the process of finding all reachable nodes by a circulating is repeated, the set of $R_0$ will be resumed by $t$ iterations. Let the set of nodes in iteration $i$, $1 \leq i \leq t$, be denoted by $V_i$. Thus, a source node (node in the first stage) in $V_i$ can access all nodes in any $V_j$, $1 \leq j \leq t$, simultaneously at the same stage in a simulated repeated pattern network. Furthermore, if a source node $b_h$ accesses a node $b_k$ in $V_i$ at a stage, then the source node $b_k$ can only access any nodes in $V_j$, $j=i+1 \mod t$, at the same stage. We call each $V_i$ a bundle and the $t$-tuple $(V_1, V_2, \ldots, V_t)$ a cluster.

All bundles in a cluster are disjoint and they have the same number of nodes (Theorem 4.15). This number is called the bundle size of the cluster. The algorithm can be used again by inputing another binary element which is not in an established cluster, and a
new cluster containing a number of bundles would be determined. We repeats this until all binary elements are put into a cluster, and the set of clusters of the link pattern of the single-stage CN is thus obtained. Any binary element can only be in a unique cluster and any element in a cluster can only access nodes in that cluster. Thus, we have the following theorem.

Theorem 4.16: The order of the single-stage circulating network group is

\[ |\langle Bx \rangle| = \prod_{i=1}^{C} |c_i| \cdot \left( \prod_{j=1}^{2|b_1|} |b_1| \right) |c_1|, \]

where

\( \{c_i | i = 1, \ldots, C\} \) is the set of clusters of \( x \), \( |c_i| \) is the number of bundles in the cluster \( c_i \), and \( |b_1| \) is the bundle size of the cluster \( c_1 \).

A cluster containing one bundle forms a small PCN by itself. A cluster containing more than one bundle forms a small strongly-connected network and does not form a PCN by itself. All inlets connected to the nodes in a bundle can achieve any permutation of themselves, and they can be taken together to another bundle in the same cluster. However, once a bundle is fixed, other bundles must be restricted to fixed positions according to the order

\[ \prod_{i=1}^{C} |c_i| \cdot \left( \prod_{j=1}^{2|b_1|} |b_1| \right) |c_1|, \]

\(^1\) denotes the product of multiplications.
in the t-tuple of the cluster specification. There is no interaction between two clusters.

The realizability of a given permutation relies then on the formation of the clusters for the link patterns. Thus, we have the following since the cluster formation can be done in polynomial time as discussed earlier.

**Theorem 4.17:** Given the link pattern of a single-stage circulating network of size n, it is easy to determine if the CN can realize a given permutation P of degree n.

As an example, the set of clusters of the link pattern in Figure 4.7 is \{({4},{1}),({5,7},{9,11},{2,6}),({3,8,10})\} obtained by calling the cluster formation algorithm described previously. By Theorem 4.16, the number of permutations realized by the CN using Figure 4.7 as BSN is \[(2)(2!)(3)(4!)\] [6!]. The permutation P=(7, 8, 18, 21, 19, 15, 1, 2, 3, 22, 4, 17, 11, 12, 5, 6, 14, 9, 20, 6, 10, 13) is not realizable by the CN because the inlets 9, 10, 13, 14 cannot access 3, 22, 11, 12 simultaneously and the new permutation obtained by interchange 4 and 22 in P can be realized by the CN.
Figure 4.7. An Example for Clusters and Bundles
4.4.6. Multistage Circulating Networks

If an \( n \times m \) BEMIN, \( m > 1 \), is used as the BSN of a CN, we call the CN **multistage circulating network**. We rephrase the PCN determination problem for multistage CN by the following question. Given an \( n \times m \) BEMIN \( \alpha \), is there an integer \( R \) such that the repeated pattern network \( (\alpha)^r \) is rearrangeable for all \( r > R \)?

Since the CN functions by circulating data through the BSN a number of times, the single-stage CN and the multistage CN are essentially similar except that the multistage CN provides more possible paths in one circulation. It is not difficult to see that the proof of Theorem 4.13 and Theorem 4.15 can be modified in order to adapt the multistage case.

The modification of the proof Theorem 4.14 only differs in case 2. In case 2, if the conflict binary element is in a stage \( S_1 \) which is not the last stage of the BSN that simulates a RPN having full access, we need to set all the binary elements arbitrarily from stage \( S_{i+1} \) to the last stage of the BSN such that the permutation \( Q \) will be realized by the BSN and can be taken to \( Qr \) by cascading \( r \) BSNs. The remaining of the proof is similar to that of Theorem 4.14. Thus we have the following.

**Corollary 4.18:** A multistage CN is a PCN iff it has full
It has full access iff there is an inlet having full access. It is easy to decide if a multistage CN is a PCN.

It follows from the proof of Theorem 4.14 that the full access condition can actually be simplified to be that any two inlets can be routed to a common binary element in a simulated repeated pattern network. One can envision that the binary elements in a BEMIN can be reduced and the PCN property still holds. Lemma 4.1 shows that it only needs one binary element to constitute a PCN. We infer here that the tree network [Levi68] as shown in Figure 4.8 for size 16 of an instance of 8 inlets can also be used as a PCN.

The generalization from Theorem 4.17 for the multistage CN is also obvious, since it is the similar process to form the set of clusters for the multistage once all its link patterns are given.

**Corollary 4.19:** It is easy to determine if a multistage CN specified by the link patterns of its BSN can realize a given permutation $P$.

As indicated earlier, a multistage CN with less number of stages can simulate a repeated pattern network with a large number of stages. However, in general a BEMIN with more stages is more powerful than a BEMIN with less stages due to the ability of performing pipelining operations. Furthermore, a BEMIN has more
Figure 4.8. Tree Network of Size 16
stages may have more fault tolerant capability (see Chapter 5). By means of adding stages with favorable link patterns, a single-stage BEMIN can become a strongly-connected network, a full access network, and then a rearrangeable network (see Chapter 3) possibly with some fault tolerance. But any n×m BEMIN with any number of stages cannot achieve strictly-sense nonblocking if n>2.

To see this, consider a BEMIN realizing a permutation P=(p₁, ...,pₙ). Since n>2, there exists two node-disjoint paths which connect inlet i to outlet pᵢ and inlet j to outlet pⱼ (see Chapter 2). If these two connections are released, and two new connections are requested to connect inlet i to outlet pⱼ and inlet j to outlet pᵢ, then to satisfy these two connections, one must rearrange existing connection paths, or use a network with extra bypassing switching elements and links such as a Clos nonblocking network [Clos53].

4.5. The CN with a Shuffle State

In the following discussion, we consider a CN whose configuration contains a shuffle state. A shuffle (perfect shuffle, outshuffle) of degree 2k is defined as the permutation (1,k+1,2,k+2, ...,k,2k). The permutation (shuffle)^⁻¹ is called unshuffle.

Theorem 4.20: The CN with configuration {shuffle, cyclic-shift}
is a PCN if the size n of the CN is even.

The proof can be found in [Golo61]. The CN will not be a PCN if n is odd. Figure 4.9 shows an implementation of the BSN of size 8. Also note that the cyclic-shift can be implemented easily by enhancing the temporary buffer to have the rotate-one function. If this is the case, the MSN only needs to implement the shuffle and the identity.

**Theorem 4.21:** The single-stage CN using shuffle as the link pattern of its BSN is a PCN.

Proof: Let \( b_1, b_2, \ldots, b_k \) be the binary elements of the BSN of size n from top to bottom, \( k = n/2 \). If k is odd, the link pattern is a \( 2^k \)-cycle. If k is even, by running the algorithm in the proof of Theorem 4.15, we obtain \( R_0 = \{1\} \), \( R_1 = \{1, 2\}, \{1, 2, 3, 4\}, \ldots, \{1, 2, \ldots, k-2\} \), and \( R_2 = \{1, 2, \ldots, k\} \). Thus, both cases satisfy the PCN condition. Q.E.D.

The CN in Theorem 4.21 has \( 2n/2 \) states and each element is independently controlled. If the single stage is controlled by one control line, a CN with only 2 states out of the \( 2n/2 \) states can be constructed. Figure 4.10 shows a CN of size 16 with configuration \{outshuffle, inshuffle\}. It is shown in [Diac82] that exactly how many permutations can be realized by combining inshuffles and
Figure 4.9. An Implementation of Shuffle, Cyclic-Shift Network
outshuffles, and the corresponding CN will not be a PCN.

If \( n=2^N \), \( N \) being positive integer, the single-stage CN using BSN \( xB \), where the link pattern \( x \) is the shuffle of degree \( n \), is called a **shuffle-exchange network** of size \( n \). The shuffle-exchange network has been used for parallel computation on many applications [Feng81]. A shuffle-exchange network of size \( n=2^4 \) is shown as in Figure 4.10 if we allow each binary element to be individually controlled.

The capability of realizing every arbitrary permutation of the shuffle-exchange network can be easily shown by Theorem 4.21. However, the solution of the CN efficiency problem seems to be difficult. In [Ston71], an algorithm to realize any permutation in \( O(\log(n))^2 \) passes is given and in [Sieg77b] another one in \( 2(\log(n))^2 \) passes is shown. In [Park80], a new bound of \( 3(\log(n)) \) shuffle-exchange steps is provided, and in [Wu81] the Benes binary network is used to reduce the upper bound to \( 3(\log(n))-1 \). A lower bound is given by the following counterexamples. It is convenient to consider the stripped BEMIN (see Chapter 3) since it realizes the same number of permutations as the BEMIN itself.

**Theorem 4.22:** The stripped \( 2(\log(n))-1 \) stage repeated pattern network of the shuffle-exchange network cannot realize the permutation \((1,2,5,6,\ldots,n-3,n-2,x_1,\ldots,x_{n/2})\), where \((x_1,\ldots,x_{n/2})\)
Figure 4.10. An Outshuffle, Inshuffle Network
is any permutation of \((3, 4, 7, 8, \ldots, n-1, n)\).

**Proof:** Consider the connections required for inlets \(1, 2, 5, 6, \ldots, n-3, n-2\) to the corresponding outlets \(1, 2, 5, 6, \ldots, n-3, n-2\); that is the subpermutation for the inlets \((1, 2, 5, 6, \ldots, n-3, n-2)\) contained in the permutation stated in the theorem. The subset \(A\) of binary elements \(1, 2, 3, \ldots, n/4\) at stage \(\log(n)-1\) are the only elements accessible from the inlets \(1, 2, 5, 6, \ldots, n-3, n-2\) due to the perfect shuffle patterns. Similarly, the subset \(B\) of binary elements \(1, 3, 5, \ldots, n/2-1\) at stage \(\log(n)\) are the only elements accessible from the outlets \(1, 2, 3, 4, \ldots, n-3, n-2\).

Since there are only \(n/4\) links provided between stage \(\log(n)-1\) and stage \(\log(n)\) for the subset \(A\) and subset \(B\), it is not possible to accomplish the desired connections of \(n/2\) pairs of inlets and outlets. Q.E.D.

Fig. 4.11 gives an example for \(n=16\) where the required connections are shown by a portion of the BEMLN acyclic digraph. Thus, the lower bound of the solution of CN efficiency problem for shuffle-exchange network is \(2(\log(n))-1\) passes.

We define a (single stage) unshuffle-exchange network as the reverse network of the BSN of the shuffle-exchange network.

**Theorem 4.23** A single stage unshuffle-exchange network can
Figure 4.11. A Permutation not Realizable

Figure 4.12. An Unshuffle Solution
partition any permutation at the inlets into 2 subpermutations, UP and DOWN, at the outlets such that each subpermutation contains exactly one element from each subset of the partition \{\{1,2\}, \{3,4\}, \ldots, \{n-1,n\}\}.

Proof: We show this by an algorithm. For a permutation \( P=(p_1, \ldots, p_n) \), we associate it with a graph such that each \( p_i \) is represented by a vertex \( v_i \) and an edge is created for every pair \( (p_i, p_j) \) if \( p_i, p_j \) are connected to the same binary element.

Step 1. If all \( p_i \) is assigned, stop.
Step 2. Start with any vertex \( p_i \) which has not been assigned. Let \( P_x=p_i \). Assign \( P_x \) UP.
Step 3. Assign \( P_y \) DOWN where \( v_y \) is adjacent to \( v_x \).
Step 4. If \( p_y \) and \( p_i \) in the same subset of the subset of the partition \{\{1,2\}, \{3,4\}, \ldots, \{n-1,n\}\}, then go to step 1; or else assign \( p_j \) UP where \( p_j \) is in the same subset of \( p_y \). Let \( P_x=p_j \), go to step 3. Q.E.D.

Fig. 4.12 gives an example where \( n=16 \) and \( P=(3,11,12,16,2,13,8,7,14,1,15,10,4,9,5,6) \). Although the permutations given in Theorem 4.22 can be realized by adding one stage to the repeated pattern network in Theorem 4.22, these permutations are not the only permutations which cannot be realized. For example, the permutation \( (1,3,5,7,4,6,2,8) \) cannot be realized by the 4-stage stripped RPN of
size 8, although the 5-stage one can realize any permutation of degree 8 (see Chapter 3). For shuffle-exchange network of size greater than 8, no exact solution has been obtained. The NP-hard results for the SPL problem and the CN efficiency problem obtained in Section 4.3.2 also reflect some difficulty in solving this problem.

4.6. Summary

A circulating network is defined by its configuration which is the set of the realizable states of its bijection state network. The extensions and variations to circulating networks are considered, and examples of networks are given.

Four problems, namely the CN capability problem, the PCN determination problem, the shortest phase length problem, and the CN efficiency problem, are solved in general by group theory application. The first two are polynomially solvable, and the last two are NP-hard.

The use of binary element multistage interconnection networks as the bijection state network are discussed with regard to group theoretic interpretations. Both the new PCN determination problem and the new CN capability problem are solved by good characterizations. Strongly-connected networks, full access
networks, full permutation networks, and nonblocking networks are
differentiated. Elaborations are done for single-stage circulating
networks, and the results are generalized to multistage circulating
networks. In particular, the single-stage circulating group is
determined as a result of the application of the algorithm which
solves the CN capability problem.

Finally, circulating networks containing a shuffle state are
discussed. Throughout this chapter, new techniques and new concepts
regarding circulating networks as well as the corresponding
simulated multistage interconnection networks are precisely defined
and unified.
Chapter 5

IMPROVEMENTS, GENERALIZATIONS, AND EXTENSIONS OF
FAULT DIAGNOSIS AND FAULT TOLERANCE CONSIDERATIONS
FOR BINARY ELEMENT MULTISTAGE INTERCONNECTION NETWORKS

Reliability plays critical role among all design parameters in
many situations, and fault diagnosis and fault tolerance are two
essential techniques to achieve reliability. A fault diagnosis
method for a class of multistage interconnection network has been
studied in [Feng79]. The study was based on the baseline network
which can be used to represent a class of topologically equivalent
multistage interconnection networks (see Chapter 3). A general
fault model was constructed; then the detecting and locating of a
single fault and the detecting of multiple faults were explored.
This chapter does improvements and generalizations to the previous
work. Improvements apply to the number of tests, and
generalizations apply to the classes of networks given in the
previous chapters. Besides these, fault diagnosis in an
asynchronous operation environment is explored and some fault
tolerance considerations for BEMINs are given.
5.1. The Fault Model

In a BEMIN (see Chapter 1) we consider three categories of faults, in the order from high to low probability of occurrence, as link faults, control faults, and element faults. Only permanent faults are assumed for all faults. A link fault is caused by either a stuck-at-zero (s-a-0) or a stuck-at-one (s-a-1) logical fault at a link or its equivalent. Many physical faults such as shorts and opens can be modeled as link logical faults [Breu76]. A bridge fault can also be treated as a link logical fault [Agra82b]. Note that we do not exclude the possible fault in a binary element which results in a s-a-0 or s-a-1. When a link fault is located, it is understood that it happens at the link or in the binary elements connected to the link. Table 5.1 gives the faulty outputs and the tests for single link fault of a single binary element. The inlets are labeled \( x_1 \) and \( x_2 \), and outlets \( y_1 \) and \( y_2 \). The type of faults, s-a-0 or s-a-1, is denoted by superscripts in the table.

A control fault is caused by the s-a-0 or s-a-1 fault of a control line or its equivalent, and as a result, the binary element always functions at state 0 or at state 1 independent of the control \[0pfe71], [Agra82b], [Shen82]. Table 5.2 gives the faulty outputs and the tests for control faults of a single binary element.

An element fault is caused by the malfunction of the binary
Table 5.1. A Binary Element Tested for Single Link Faults

<table>
<thead>
<tr>
<th>Valid State</th>
<th>Fault</th>
<th>Test</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$x_1$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>$S_{10}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1, y_1$</td>
<td>0, 0</td>
<td>1, 0</td>
<td>1, 0</td>
</tr>
<tr>
<td></td>
<td>$x_1, y_1$</td>
<td>0, 1</td>
<td>0, 1</td>
</tr>
<tr>
<td></td>
<td>$x_2, y_2$</td>
<td>0, 1</td>
<td>0, 1</td>
</tr>
<tr>
<td></td>
<td>$x_2, y_2$</td>
<td>1, 0</td>
<td>1, 0</td>
</tr>
<tr>
<td>$S_{5}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1, y_2$</td>
<td>0, 0</td>
<td>1, 0</td>
<td>1, 0</td>
</tr>
<tr>
<td></td>
<td>$x_1, y_2$</td>
<td>0, 1</td>
<td>0, 1</td>
</tr>
<tr>
<td></td>
<td>$x_2, y_1$</td>
<td>0, 1</td>
<td>1, 1</td>
</tr>
<tr>
<td></td>
<td>$x_2, y_1$</td>
<td>1, 0</td>
<td>0, 1</td>
</tr>
</tbody>
</table>
Table 5.2. A Binary Element Tested for Single Control Faults

<table>
<thead>
<tr>
<th>Valid State</th>
<th>Fault</th>
<th>Test</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_1$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>s-a-0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$(s_{10},s_{10})$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>s-a-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$(s_5,s_5)$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
element. Table 5.3 lists the 16 possible "states" of the binary element [Feng79] in which states $s_5$ (state 1) and $s_{10}$ (state 0) are valid states. An element fault may be a deviation from a $s_5$ or $s_{10}$ or both. For example, $(s_{10}-s_2, s_5-s_1)$ means that the binary element functions at state $s_2$ when the setting should be $s_{10}$, and it functions at state $s_1$ when the setting should be $s_5$. We write $(s_2, s_1)$ for short. Thus, there are 255 faults and only $(s_{10}, s_5)$ is considered to be functionally normal. Table 5.4 and Table 5.5 give the faulty outputs and the tests for a single binary element functioning at $s_{10}$ and $s_5$, respectively, where "-" means the logically undefined output (e.g. floating voltage) and "$\phi$" means logically erroneous output where 0 and 1 are the simultaneous inputs (e.g. merging voltage). Thus, the element fault contains parametric faults [Breu76]. Note that - may be generated by the faulty states $s_0$, $s_1$, $s_2$, $s_4$, $s_6$, $s_8$, $s_9$ and the faulty output is independent of the inputs, while the faulty output "$\phi$" depends on the values of inputs and it may be generated by the faulty states $s_6$, $s_7$, $s_9$, $s_{11}$, $s_{13}$, $s_{14}$, and $s_{15}$. It is assumed that - and $\phi$ can be differentiated from each other and from 0 and 1 during the test. We observe that only faults' $(x,y)$, $x,y=s_3$, $s_5$, $s_{10}$, or $s_{12}$, $(x,y)\neq(s_{10},s_5)$ are logical faults. We call these faults element logical faults.

In this chapter, it is assumed that individual binary element is not accessible while testing a BEMIN. In the discussion, we also
Table 5.3. The 16 Possible States of a Binary Element

<table>
<thead>
<tr>
<th>State Name</th>
<th>Switching Element Symbol</th>
<th>Crosspoint Switching Matrix Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>![State 0 Diagram]</td>
<td>$(0000)$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>![State 1 Diagram]</td>
<td>$(0001)$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>![State 2 Diagram]</td>
<td>$(0010)$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>![State 3 Diagram]</td>
<td>$(0011)$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>![State 4 Diagram]</td>
<td>$(0100)$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>![State 5 Diagram]</td>
<td>$(0101)$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>![State 6 Diagram]</td>
<td>$(0110)$</td>
</tr>
<tr>
<td>$s_7$</td>
<td>![State 7 Diagram]</td>
<td>$(0111)$</td>
</tr>
<tr>
<td>$s_8$</td>
<td>![State 8 Diagram]</td>
<td>$(1000)$</td>
</tr>
<tr>
<td>$s_9$</td>
<td>![State 9 Diagram]</td>
<td>$(1001)$</td>
</tr>
<tr>
<td>$s_{10}$</td>
<td>![State 10 Diagram]</td>
<td>$(1010)$</td>
</tr>
<tr>
<td>$s_{11}$</td>
<td>![State 11 Diagram]</td>
<td>$(1011)$</td>
</tr>
<tr>
<td>$s_{12}$</td>
<td>![State 12 Diagram]</td>
<td>$(1100)$</td>
</tr>
<tr>
<td>$s_{13}$</td>
<td>![State 13 Diagram]</td>
<td>$(1101)$</td>
</tr>
<tr>
<td>$s_{14}$</td>
<td>![State 14 Diagram]</td>
<td>$(1110)$</td>
</tr>
<tr>
<td>$s_{15}$</td>
<td>![State 15 Diagram]</td>
<td>$(1111)$</td>
</tr>
<tr>
<td>Fault</td>
<td>Test</td>
<td>Normal</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>$S_{10}-S_0$</td>
<td>$x_1 \times x_2$</td>
<td>$y_1 \times y_2$</td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>0 1</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0</td>
<td>1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_1$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_2$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_3$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_4$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_5$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_6$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_7$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_8$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_9$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_{10}$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_{11}$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_{12}$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_{13}$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_{14}$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>$S_{10}-S_{15}$</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>
Table 5.5. A Binary Element Tested for $s_5$-Malfunctions

<table>
<thead>
<tr>
<th>Fault</th>
<th>Test $x_1$ $x_2$</th>
<th>Output $y_1$ $y_2$</th>
<th>Normal $y_1$ $y_2$</th>
<th>Faulty $y_1$ $y_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_5-s_0$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>- -</td>
</tr>
<tr>
<td>$s_5-s_1$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_2$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>- 1</td>
</tr>
<tr>
<td>$s_5-s_3$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_4$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_5$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_6$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_7$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_8$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_9$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_{10}$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_{11}$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_{12}$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_{13}$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_{14}$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
<tr>
<td>$s_5-s_{15}$</td>
<td>0 1 1 0 0 1 1</td>
<td>1 0 0 1 1 1 1</td>
<td>1 0</td>
<td>1 -</td>
</tr>
</tbody>
</table>
classify five levels of faults. A **single link fault** is the situation where only a link fault occurs in a network. A **single line fault** is either a link fault or a control fault. A **single logical fault** is a link fault, a control fault, or an element logical fault. A **single fault** is a link fault, a control fault, or an element fault. A **multiple fault** contains at least two faults of link, control, or element faults. The applicability of the models depend on the technology and implementations of the network. When situations are more optimistic, a lower level model can be used.

A **test** is an application of one bit (0 or 1) to each desired inlet as input together with the examination of the outputs at the desired outlets. It is assumed that all faults occurring in the network will propagate to the outlets. Although in general the application of "-" and "∅" as inputs may reduce the number of tests, we do not consider this method when the main concern is logical faults in this chapter.

### 5.2. Detecting Single Faults

According to the fault model, a fault in a BEMIN can be a link fault, a control fault, or an element fault. For detection purposes, there are only two classes of faults as stated below.

**Lemma 5.1:** A control fault is equivalent to either a \((s_{10}, s_{10})\)
element fault or a \((s_5, s_5)\) element fault.

A link fault can be either a stuck-at-0 or a stuck-at-1. A binary element fault can be considered as the malfunction of the binary element from its valid states. The following theorem gives the minimum number of tests for detecting single faults.

**Theorem 5.2:** Three tests are necessary and sufficient for detecting single faults in a baseline network constructed of binary elements with two valid states \(s_{10}\) and \(s_5\).

**Proof:** Consider one binary element with inputs \(x_1, x_2\) and outputs \(y_1, y_2\) first. To detect a single fault we need at least two tests, one for binary element at state \(s_{10}\) and the other at \(s_5\). From Table 5.4 and Table 5.5 it can be seen that the test \((x_1, x_2) = (0, 1)\) or \((1, 0)\) can detect all types of \(s_{10}\)-malfunction and/or \(s_5\)-malfunction, but the test \((x_1, x_2) = (1, 1)\) or \((0, 0)\) cannot. However, any combination of the two tests \([(0, 1)\) for both \(s_{10}\) and \(s_5\), \((1, 0)\) for both \(s_{10}\) and \(s_5\), \((0, 1)\) for \(s_{10}\) and \((1, 0)\) for \(s_5\), or \((1, 0)\) for \(s_{10}\) and \((0, 1)\) for \(s_5)\) is not sufficient to detect all the link faults. Thus, at least three tests are required. Let \((0, 1)\) and \((1, 0)\) tests be used for binary element functioning at \(s_{10}\) so that any single link fault or the \(s_{10}\) malfunction can be detected. Then let \((0, 1)\) test be used for the binary element functioning at state \(s_5\) to detect the \(s_5\)-malfunction. Thus, totally three tests.
are necessary and sufficient to detect single faults for a two-state binary element.

For a baseline network with more than one element, it is shown [Feng79] that if the inlets, labeled in binary numbers, with even or odd number of 1's receive logical 0 or 1 respectively, then any element in the network will have the input (0, 1) or (1, 0) when the network functions normally with all its elements at state $s_{10}$ or $s_5$. Thus, establishing three such tests is necessary and sufficient to detect single faults in a baseline network. Q.E.D.

**Theorem 5.3:** Three tests are necessary and sufficient to detect single faults in any BEMIN.

**Proof:** The first test for the BEMIN is any input test such that any two inlets connected to the same first stage binary element receive two different values 0 and 1, one for each. We claim that there is a setting of the network such that every binary element receives a 0 on one of its two input terminals, and a 1 on the other input terminal.

Suppose that the inputs of stage $S_1$ are such that every binary element in $S_1$ has $(0,1)$ or $(1,0)$ inputs. Let the link pattern $F_1$ be $(p_1, \ldots, p_n)$. It is not difficult to see that for any $2^k$-cycle (see Chapter 2) in the link pattern, we can set the corresponding binary
element in $S_i$ such that every binary element of stage $S_{i+1}$ in the cycle receives either $(0,1)$ or $(1,0)$ input. Since the link pattern contains only $2^k$-cycles, all binary elements in $S_{i+1}$ has $(0,1)$ or $(1,0)$ input.

The second test is applied by complementing all the inputs and uses the same setting of the network determined in the application of the first test. Consequently, every binary element and every link now receive the complementary inputs as that of the first test. The third test uses the same inputs as in the second test but the setting of the network is such that the state of every binary element is complemented ($s_5$ becomes $s_{10}$ and vice versa).

Thus, each binary element is tested at its two valid states with $(0,1)$ or $(1,0)$ inputs and the link fault is also determined by the combination of the first and the second test. Q.E.D.

The above theorem also settles the problem of determining the minimum number of tests needed to detect a single fault, given an BEMIN, to be easy (constant-time solvable).
5.3. Locating Single Link Faults

In this section, we show that the test set defined in the previous section can be used to locate single link stuck fault in a baseline network. The locating process will be conducted by two tests in the first phase and one test in the second phase. See Figure 5.1. During phase 1, the inlets, labeled in binary numbers, with even or odd number of 1's receive input vector 01 or 10 (alternatively, 10 or 01), respectively. Based on the result of phase 1 test, all inlets then receive either all 1's or all 0's during phase 2 test. Figure 5.2 shows an alternate test scheme.

Theorem 5.4: Independent of network sizes, three tests are necessary and sufficient for detecting and locating single link faults in a baseline network constructed of switching elements with two valid states $s_{10}$ and $s_{5}$.

Proof: The necessary condition is obvious because it requires at least two tests (phase 1) in order to detect the link faults and at least one additional test to locate the fault. The sufficient condition can be proved due to the fact that during phase 1 test the type of link stuck fault is determined and a unique faulty path can be computed between the faulty output and its input [Feng79]. Thus, only one subsequent test is required to determine the other faulty path during phase 2 so that the intersection of these two paths
Figure 5.1. Fault-Free Response Scheme for Single Link Faults
Figure 5.2. An Alternate Test Scheme
gives the faulty link. Q.E.D.

Figure 5.3 gives an example of the detection and location of link faults. Since phase 1 test identifies the link fault to be a s-a-0 type, every input terminal then receives a 1 during the phase 2 test. From these two tests the possible faulty links are identified to be (6, 6, 3, 5, 6) for phase 1 and (7, 6, 2, 0, 1) for phase 2. Intersecting these two sets we find that the link s-a-0 fault is located at link 6 between stage 1 and stage 2.

Another test set which contains also three tests can be applied to detect and locate single link faults. Again, there are two tests in the first phase and one test in the second phase. The first two tests which are inputing all 0's and inputing all 1's at all input terminals can decide the link fault to be a stuck-at-0 or a stuck-at-1. Then the third test will input all 1's providing a stuck-at-0 fault or all 0's providing a stuck-at-1 fault. The fault patterns generated are sufficient to locate the link stuck fault.

Without further techniques, the above tests can directly be used to detect and locate single link fault in a baseline network with binary elements having two additional states s12 and s3. The reason is obvious since we only adopt states s10 and s5 to diagnose link fault in the network.
Figure 5.3. An Example of Locating the Link Fault
We generalize the result of Theorem 5.4 to partial transversal networks and minimum full access networks (see Chapter 3).

**Theorem 5.5:** Three tests are necessary and sufficient for detecting and locating single link faults in any PTN, PTN⁻¹ or MFA network.

**Proof:** The first two tests determine a faulty path containing a potential faulty link. By complementing the states of the binary elements along the path plus the setting of other binary elements in the network, we need to prove that any inlet/outlet connection path contains at most one potential faulty link in this third test.

Firstly, we prove for the GBBN case and the generalization to PTN and PTN⁻¹ follows obviously from the definitions and the arguments. Let the faulty path $a_0a_1...a_m$, $m=2(\log(n))-1$, be a path routed through GBBNₜ, i.e., the link $a_1$ is led into GBBNₜ and the link $a_{m-1}$ is led out of GBBNₜ.

By complementing the binary element which $a_0$ and $a_1$ are connected to, a new path is led into GBBNₜ from links $a_0, b_1$. By complementing the binary element which $a_m$ and $a_{m-1}$ are connected to, a new path is led out of GBBNₜ to links $b_{m-1}, a_m$. It is not difficult to choose a setting for GBBNₜ such that $b_1$ is not connected to $b_{m-1}$. Thus, the path $a_0b_1...x_m$ contains only one
potential link $a_0$, and the path $y_0...b_{m-1}a_m$ also contains only one potential link $a_m$. The remainder of the proof is based on the recursive construction of GBBN and it is omitted for simplicity.

Secondly, we use the property 5 of Corollary 3.26 to prove the MFA network case. Let $(a_i,a_{i+1})$ be the binary element to which the links $a_i$ and $a_{i+1}$ is connected along the faulty path $a_0a_1...a_m$. Since the states of all the binary elements along $a_0a_1...a_m$ are complemented, we claim that an inlet/outlet pair used to check the link $a_i$ does not pass through any links $a_j$, $0<j<m$, $j\neq i$.

Let the roots of the two immediate descendant subtrees of the left-tree of $(a_i-1,a_i)$ be $(a_i-2,a_{i-1})$ and $(b_i-2,b_{i-1})$. Let the roots of the two immediate descendant subtrees of the right-tree of $(a_i,a_{i+1})$ be $(a_{i+1},a_{i+2})$ and $(b_{i+1},b_{i+2})$. The checking paths must be led from an inlet in the left-tree of $(b_i-2,b_{i-1})$, to the link $b_{i-1}$, passing through the link $a_i$, to the link $b_{i+1}$, and then into the right-tree of $(b_i+1,b_{i+2})$ which contains a desired outlet. Since the links $a_0,...,a_{i-2}$ are in the left-tree of $(a_{i-2},a_{i-1})$, the links $a_{i+2},...,a_m$ are in the right-tree of $(a_{i+1},a_{i+2})$, and $a_{i-1}\neq b_{i-1}$ and $a_{i+1}\neq b_{i+1}$, the claim is correct. Q.E.D.

It should be pointed out that three tests are not sufficient to locate single faults if the link patterns of a network contain a 2*1-cycle for obvious reasons.
5.4. Locating Single Faults

The first level fault model is studied in the previous section. In this section, we investigate the locating of single control faults and single elements faults and study fault diagnosis in the rest of the levels in the fault model. By combining the minimum number of tests obtained in each case, we give a procedure to diagnose single logical faults for any arbitrary minimum full access network.

5.4.1. Single Control Faults

A control fault is a stuck fault occurred at a control line such that the binary element is stuck at one of its valid states. From Lemma 5.1, there are two types of control faults indistinguishable from two element logical faults and can be denoted by \((s_5,s_5)\) (stuck-at-1) and \((s_10,s_10)\) (stuck-at-0).

Theorem 5.6: For any MFA network, single control fault can be detected and located by two tests and these faults are distinguishable from single link faults.

Proof: From the proof of Theorem 5.3, we know that there is a normal setting of the network such that each binary element receives either a \((0,1)\) or a \((1,0)\) input. The normal output pattern is uniquely determined once the inputs are applied for this first test.
Let a binary element be set at a state $s_{10}$ as normal. If it is stuck at $s_5$, the faulty output pattern can be observed due to the fact that there is an interchange of positions between a 0 and a 1. It is similar, if $s_5$ is the normal state for the binary element.

Since an MFA has the property that there is a single path to connect any inlet to any outlet and any two paths meet (intersect) at most one binary element when the setting of the network is decided for all binary elements (see Corollary 3.26), no two interchanges corresponding to control struck faults of two binary elements apply to the two outputs at the same two outlets. Thus, any single control fault which is a deviation from the normal setting can be located by checking the faulty output pattern at this test.

At the second test, all the states of the binary elements are complemented. The same argument as for the first test holds and each binary element has been tested for its two normal states $s_{10}$ and $s_5$.

Since any single link fault during either test will give a faulty output pattern with unequal number of 0's and 1's at the outlets, any single control fault can be discriminated from any single link fault. Q.E.D.
The above proof implies the determination of single line faults as below.

**Corollary 5.7:** The minimum number of tests for detecting and locating single line faults is three for any MFA network.

### 5.4.2. Single Element Logical Faults

There are 15 types of single element logical faults, namely, \((x,y), x, y = s_3, s_5, s_{10},\) or \(s_{12},\) and \((x,y) 
\neq (s_{10}, s_5).\) Among these, we firstly exclude the faulty types having \(s_i - s_5\) or \(s_j - s_{10}\) malfunctions, \(i = 3, 10,\) or \(12,\) and \(j = 3, 5,\) or \(12,\) since a binary element with one of these faults can be detected and located easily by the method given in the previous subsections and the faulty types can be determined easily by the arguments in this subsection.

If every binary element of an MFA network receives \((0,1)\) or \((1, 0)\) input for the normal function of the network during the first test, a faulty \(s_3\) or \(s_{12}\) state of a binary element in stage \(S_j\) will have the same faulty output pattern as if a link stuck fault occurred in the link pattern \(F_j\) for one of the links connected to the binary element.

Suppose that a faulty \(s_3\) or \(s_{12}\) state exists. The unique faulty output at an outlet will allow us to trace back from the outlet side to the inlet side such that the potential faulty binary
element can be decided along a single path from a first stage binary element to the faulty outlet. Furthermore, the faulty state being either \( s_3 \) or \( s_{12} \) can also be determined once a potential faulty binary element is faulty in fact.

We exemplify this by Figure 5.4. The network is tested assuming it is fault-free. The change at the outlet from 0 to 1 is observed and a faulty binary element is assumed. We trace from the outlet side to the inlet side following the input terminals with 0 as input. The faulty path is arrowed and the possible faulty states are denoted under the potential faulty binary elements.

In order to test the malfunction of a binary element from the other valid state, the second test is applied by complementing all the states of all binary elements. There are two cases.

Case 1. Only one of the tests detects a fault. The fault could be \((s_3,s_5), (s_{12},s_5), (s_{10},s_3) \) or \((s_{10},s_{12})\). Case 2. Both tests detect a fault. The fault could be \((s_3,s_3), (s_3,s_{12}), (s_{12},s_3), \) or \((s_{12},s_{12})\).

In case 1, to determine the faulty location among the \( \log(n) \) binary elements, a binary search approach is used. For each additional test, change half of the potential faulty binary elements to the other valid state, which will function normally. Depending on whether the fault disappears or not, the potential faulty
Figure 5.4. Diagnosis Case 1 of Single Element Logical Faults
elements are reduced to the half with states changed or the other half. Thus, \(^{\log(\log(n))}\) additional tests are required.\(^1\) If a sequential search approach is used, \(\log(n)-1\) additional tests may be needed for the worst case, but the faulty element may be located by less than \(^{\log(\log(n))}\) tests.

In case 2, we claim that there are at most 2 potential faulty elements which are adjacent to each other. To prove this, suppose the first test decides a path, path 1, containing \(\log(n)\) potential faulty elements, one in each stage. Assuming that path 1 passes every binary element through the input terminal with 0 input does not lose generality. The second test is applied by complementing all the states of all binary elements. Thus, all the inputs to stage \(S_i\), \(i\) odd, remain the same and all the inputs to stage \(S_j\), \(j\) even, are complemented under the assumption that the network is fault-free.

The faulty path in the second path must also pass every binary element through either the input terminal with 0 input or the input terminal with 1 input. If this path, path 2, is marked during the first test, it passes binary elements through input terminals with alternate 0 and 1 inputs. Consider the first test and let \(b_{ij}\) be

\(1^x\) denotes the smallest integer greater than or equal to \(x\).
the right most binary element which the two faulty paths intersect. If path 2 is led into $b_{ij}$ from the outlet side with a terminal labeled 1, it must be led by the terminal labeled 0 into a $b_{(i-1)h}$ which is on path 1, except the case when $b_{ij}$ is a first stage binary element. Then path 2 must be led by the terminal labeled 1 into a $b_{(i-2)k}$. The MFA network has the property that path 1 and path 2 will not intersect again (see Corollary 3.26). If path 2 is led into $b_{ij}$ with a terminal labeled 0 and $b_{ij}$ is an element at the last stage, it must be led into a $b_{(i-1)h}$ by a terminal labeled 1. In this case, the two paths only intersect at most one binary element.

To decide which element is faulty out of the two potential ones, we apply one additional test which is the same as the first test except that the input at an inlet is complemented such that the right potential faulty element receives (0,0) or (1,1) input and its fault will be masked if it is the faulty element. If the test examines a faulty output pattern, then the left potential faulty element is indeed the faulty one; otherwise, it is fault-free and the right element is the faulty element. The type of the fault is determined earlier by the faulty path.

Figure 5.5 gives an example. The potential fault can be $(s_3, s_{12})$ at the top-left element or $(s_3, s_3)$ at the top-right element. The third test discriminates that the top-left one is faulty by
observing a faulty output pattern.

An \((s_5, s_{10})\) fault can be easily detected and located during the first test which has the same output pattern as the \((s_5, s_5)\) fault case. These two types are distinguished by the second test. The case of \((s_{10}, s_{10})\) fault is similar. It is not difficult to see that \((s_3, s_{10}), (s_5, s_3), (s_{12}, s_{10}),\) and \((s_5, s_{12})\) faults can be determined easily by combining the methods presented in this and the previous subsection.

It can be observed from Table 5.1, Table 5.4 and Table 5.5 that if \((x_1, x_2) = (0, 0)\) or \((1, 1)\) is applied to the inlets, none of the output caused by the binary element malfunction has the same pattern as that generated by a link fault.

By inputing all 1's to the network, the output pattern will have a single 0 at one terminal with 1's at the rest of the terminals if the network has a single stuck-at-0 fault and each binary element is fixed to a particular state. Any single binary element in the network having a malfunction will not generate the same output pattern. Thus, the single stuck-at-0 fault is distinguishable from single element faults. By inputing all 0's to the network, it is similarly shown that the single stuck-at-1 fault can be distinguished from single element faults.
The first test

The second test

The third test

Figure 5.5. Diagnosis Case 2 of Single Element Logical Faults
Concluding the discussion in this subsection, we have the following theorem.

**Theorem 5.8:** Single element logical faults can be detected and located by $2+\log(\log(n))$ tests for any MFA network of size $n$, and these faults are distinguishable from single link faults.

### 5.4.3. A Diagnosis Procedure for Single Logical Faults

As noted earlier, there are single logical faults which cannot be pinpointed down to be either control faults or element logical faults. Except these $(s_5,s_5)$ and $(s_{10},s_{10})$ two faults, all single logical faults are uniquely determined. The following "branch-and-bound" procedure is based on the theorems presented in previous sections for detecting and locating single logical faults in an MFA network.

1. **Step 1.** Let every element receive $(0,1)$ or $(1,0)$ input. If there is no fault, go to step 5.

2. **Step 2.** Locate the faulty binary element for $(s_5,s_5)$, $(s_5,s_{10})$, $(s_{5},s_{3})$, or $(s_{5},s_{12})$ fault and go to step 7. Complement all states of all binary elements and use all 1's or all 0's inputs for all inlets. If there is no fault, go to step 3. Locate the faulty link and halt the procedure.
Step 3. Use the inputs as in step 1. If there is no fault, go to step 4. If it is \((s_3,s_{10})\) or \((s_{12},s_{10})\) fault, locate it and determine the faulty type. Locate the faulty element for \((s_3,s_3)\), \((s_3,s_{12})\), \((s_{12},s_3)\), or \((s_{12},s_{12})\) fault by at most one additional test and decide its faulty type. Halt the procedure.

Step 4. Use binary search to locate either \((s_3,s_5)\) or \((s_{12},s_5)\) fault by "\(\log(\log(n))\)" tests and determine its faulty type. Halt the procedure.

Step 5. Complement all inputs. If there is no fault, go to step 7.

Step 6. Complement all states of all binary elements and use all 1's or all 0's inputs for all inlets. Locate the faulty link and halt the procedure.

Step 7. Complement all states of all binary elements. If there is no fault, go to step 9. If it is \((s_{10},s_3)\) or \((s_{10},s_{12})\) fault, go to step 8. Locate \((s_{10},s_{10})\) fault or decide the faulty type to be \((s_5,s_{10}), (s_5,s_3), (s_{12},s_{10}),\) and halt the procedure.

Step 8. Use binary search to locate either \((s_{10},s_3)\) or \((s_{10},s_{12})\) fault by "\(\log(\log(n))\)" tests and determine its faulty type. Halt the procedure.
Step 9. Decide whether the faulty type is \((s_5, s_5)\) or there is no fault in the network. Halt the procedure.

As indicated in [Feng79], there are single element faults which cannot be pinpointed down to one binary element because of the existence of the "-" fault. Figure 5.6 gives an example, where the dashlines indicate the potential faults and all four possible state combinations for the two potential faulty elements are shown. Except this, the technique used previously can be similarly applied to diagnose single faults. We do not detail this because our main concern is logical faults here.

5.5. Multiple Faults

In [Feng79], an upper bound to detect multiple faults is given by \(2(l+\log(n))\) tests. This number can be reduced to \(2(l+!\log_4 n!\) if one uses also "-" and "\(\Phi\)" as inputs.\(^1\)

To detect multiple link faults, two tests are sufficient by using all 0's and all 1's inputs for an arbitrary setting of the network. In general, the faulty multiple links cannot be located. For example, if all 0's appear always at the outputs, it is not decidable where the stuck-at-0 faults occur in a link pattern or

\(^1\)!x! denotes the largest integer less than or equal to \(x\).
Figure 5.6. The Existence of '->' Faults
several link patterns.

Since each setting of an MFA network corresponds to a different permutation of the inlets at the outlets, the multiple stuck-at-$s_5$ faults can be detected and located by applying $n$ distinctive vectors to the inputs for all binary elements set at state $s_{10}$. A second test can be applied for all binary elements set at state $s_5$. Thus, we have the following.

Theorem 5.9: Multiple control faults can be detected and located by $2(\log(n))$ tests for any MFA network.

5.6. Asynchronous Fault Diagnosis

The above discussion assumes that the entire network is available for fault diagnosis at one time. When fault diagnosis is applied, the whole network must be taken offline. This happens in a SIMD environment. In another situation such as MIMD or even distributed processing environment, calls (connections) are set up asynchronously. It is economic to enforce fault diagnosis for some idle pairs of inlets and outlets while other connections are passing messages. Once a fault is detected and located, the redundancy built into the network can bypasses the fault and achieves fault tolerance capability.
5.6.1. Single Link Faults

As indicated previously, in an MFA network, a single link fault for any idle inlet/outlet pair can be detected by two bits, 0 and 1, and the type of faults can also be determined to be either \( s\text{-}a\text{-}0 \) or \( s\text{-}a\text{-}1 \). The problem is then to find where the fault locates along the faulty path.

We define a **probe** to be the process of setting necessary binary elements in order to connect a desired inlet/outlet pair.

**Theorem 5.10:** For any \( n \times m \) BEMIN, single link fault can be located in at most \( m \) probes. At each probe, at most one pair of inlets and outlets is affected in a one-bit time.

**Proof:** Let the faulty path of links be labeled \( a_0, a_1, \ldots, a_m \) from left to right, which connects inlet \( x = a_0 \) to outlet \( y = a_m \). It does not lose generality if we assume that the fault is a \( s\text{-}a\text{-}0 \).

The locating process has \( m \) probes. In probe \( i, 1 \leq i \leq m \), we test whether the fault is at location \( a_k \), \( k=i-1 \), by sending a logical 1 from \( x \) along the path \( a_0, a_1, \ldots, a_k, \ldots \) to an outlet \( y_k \). It will be shown that if \( y_k \) is idle, then no active terminals will be affected, and if \( y_k \) is active, then only \( y_k \) and the inlet \( x_i \) which is being connected to \( y_k \) will be affected.
Let \((a_k, a_{k+1})\) denote the binary element which links \(a_k\) and \(a_{k+1}\) are connected to. Let \(b_k\) and \(b_{k+1}\) be the other two links connected to \((a_k, a_{k+1})\). Thus, the binary element \((a_k, a_{k+1})\) is also named by \((b_k, b_{k+1}), (a_k, b_{k+1}),\) or \((b_k, a_{k+1})\).

If \(a_0, a_1, \ldots, a_m\) are without fault, then the faulty link is \(a_m\). Suppose that \(a_0, a_1, \ldots, a_{k-1}\) are fault-free, \(0 \leq k \leq m\). To determine \(y_k\) checking if link \(a_k\) is faulty, we check if link \(b_k\) is active. If it is, the outlet connected to it through this active path will be \(y_k\). If \(b_k\) is idle, then there must be an idle inlet/outlet pair connected through the links \(b_k\) and \(b_{k+1}\) in any BEMIN. The idle outlet will be \(y_k\). The test path is then \(a_0, a_1, \ldots, a_k, b_{k+1}, \ldots, b_m = y_k\) in order to determine if the fault is at \(a_k\), and it is assumed that \(b_{k+1}, \ldots, b_m\) are fault-free. Q.E.D.

We locate the fault in a left to right fashion above in the proof. An alternative method can be used to reduce the probability of the number of affected terminals. This process approaches the faulty link from both the left-hand and the right-hand side of the network. At a probe of the process, suppose that \(a_0, a_1, \ldots, a_i\) and \(a_{m-j}, a_{m-j+1}, \ldots, a_m\) are links without fault. The next probe will test link \(a_{i+1}\) if \(a_{i+1}\) does not affect active terminals; otherwise, the link \(a_{m-j-1}\) will be tested.

For an MFA network, the probability of the number of affected
terminals can be further reduced since each link $a_l$ along the faulty path can be probed independently of the other links of the path (see Theorem 5.5). Thus, one can use a strategy to first probe those $a_i$'s which will not affect active terminals. To probe a link $a_l$, there are no terminals affected if and only if the binary element $(a_{l-1},a_l)$, as long as existing, or any binary element on the "left extension" of $(a_{l-1},a_l)$, and the binary element $(a_l,a_{l+1})$, as long as existing, or any binary element on the "right extension" of $(a_l,a_{l+1})$, are idle. By left extension of $(a_{l-1},a_l)$ we mean that the including of binary elements $(a_{l-2},a_{l-1}),(a_{l-3},a_{l-2}),\ldots,(a_{l-j},a_l)$, all $a_{i-h}$, $1 \leq h \leq j$, being known to be fault-free, along the path in the direction from $(a_{l-1},a_l)$ to the inlet side. The right extension is similarly defined.

For example, in Figure 5.7, a faulty path in an MFA network is shown, and links $a_5$, $a_9$, $a_{10}$, and $a_{11}$ can be tested at the same time without affecting any active terminals, assuming that $a_3$ and $a_4$ are fault-free links.

If the faulty link must be located immediately, we can use the following corollary which follows from Theorem 5.5.

**Corollary 5.11:** For any MFA network of size $n$, single link fault can be located by one test, and the number of pairs of inlet/outlet affected is equal to the number of active binary
Figure 5.7. Locating Single Link Faults Asynchronously
elements along the faulty path, which is at most $\log(n)$.

Thus, there is a trade-off between the saving of the time to complete the locating processing and the reduction of terminals affected.

5.6.2. Single Logical Faults

Since the network is operated in a centralized control, circuit switching mode, any requested inlet/outlet connection path should be tested to assure fault-free before it is used as an active path to pass messages. To test the path, all the binary elements along the path must be tested at the proper state either $s_{10}$ or $s_5$. We assume that an active path remains fault-free until it becomes idle again. Thus, every binary element along an active path is normal at the state it is set, and it is called an active binary element. An idle binary element is assumed to be at state $s_0$. The valid states of a binary element therefore are $s_0$, $s_5$, and $s_{10}$.

When a connection of an idle inlet/outlet pair is requested, we diagnose the network for single logical faults along the desired path. A cycle of single logical fault diagnosis is completed when all links are used at least once and all binary elements have become active for both states $s_{10}$ and $s_5$ at least once. Thus, the length of a cycle depends on the pattern of requests.
For fault diagnosis of single logical fault, consider that an idle pair of inlet/outlet is requested for connection when some active paths have been set up. We assume that this request can be satisfied if the network is fault-free.

We set the binary elements along the desired path at the desired states, and send a vector of two bits, 01, through the path. If there is no output received, the single fault occurs at a binary element along the path. This case will be discussed later. If two bits are received at the outlet, there are three kinds of the value. If the value is 00 or 11, locate the faulty link by using the technique presented in the previous subsection. If the value is 01, there might be some binary element functions at a faulty state $s_3$ or $s_12$. For any questionable elements, i.e. idle binary elements, we know it is $s_3$ or $s_12$ if it indeed is faulty (see Section 5.4.2). In order to test a questionable element, we set up another new path which passes through the questionable element via another pair of input/output terminals. Since the questionable element is idle before, the new path connects an idle pair of inlet/outlet.

We send a vector of two bits, 10, through the new path. If there is no output received, the single fault occurs at a binary element along the new path. If two bits are received at the outlet, there are four kinds of the value. If the value is 00 or 11, locate
the faulty link along the new path by using the technique presented in the previous subsection and the old path is fault-free. If the value is 10, the questionable element functions actually normal at the desired state. If the value is 01, the fault is caused by the questionable element being \( s_3 \) or \( s_{12} \), which is known earlier.

For the no output case, the following two theorems will locate the faulty element.

**Theorem 5.12:** For any MFA network, single control fault can be located by a one-bit test and no active inlets or outlets are affected.

**Proof:** Let links \( a_1a_2...a_m \) be the potential faulty path which passes through binary elements with a single control fault at one of the idle binary elements. Put a logical 0 (or 1) at \( a_1 \), and there is no output at \( a_m \).

For each idle binary element \( (a_1,a_{i+1}) \), set the binary elements \( (b_{i+1},b_{i+2}),(b_{i+3},b_{i+4}),... \), and \( (b_{m-1},b_m) \) along the idle links \( b_{i+1},b_{i+2},...,b_m \) such that if and only if \( (a_1,a_{i+1}) \) is the faulty element, \( b_m \) will receive the value. Since the element \( (a_1,a_{i+1}) \) is idle before the test, \( b_m \) is idle and does not receive any output in the normal situation. Thus, the outlet which receives a value 0 tells the location of the control fault since all established paths
branching out from the faulty path are disjoint. Q.E.D.

**Theorem 5.13:** For any MFA network of size $n$, single element logical fault can be located by a vector test of $\log(k+1)$ bits, where $k$ is the number of potential faulty binary elements, $k<\log(n)$, and no active inlets or outlets are affected.

**Proof:** Any idle binary element along the faulty path $a_1a_2...a_m$ is a potential faulty element. A logical faulty element will function at state $s_3$, $s_12$ or $s_5(s_{10})$ if the normal state is $s_{10}$ (or $s_5$, respectively). From previous discussion, it is known that the faulty be $s_3$ or $s_12$ by tracing along the faulty path, if the faulty is $s_3$ or $s_12$.

For each idle binary element $(a_i,a_{i+1})$, we set necessary idle binary elements such that a path $b_1,b_2,..., b_i,b_{i+1},...,b_m$ passes through $(a_i,a_{i+1})$ by the other idle elements $b_i$ and $b_{i+1}$, and $b_{m}=x_i$ is an idle inlet and $b_{m}=y_i$ is an idle outlet. For all $x_i$'s and $a_1$ we put different input values, one for each. Thus, for $k+1$ values, each inlet receives a vector of $\log(k+1)$ bits. From Corollary 3.26, we obtain the following.

All $y_i$'s have value equal to $x_i$ and $a_m$ has the value equal to $a_1$ iff there is no fault. For a binary element having a $s_5$-$s_{10}$ or $s_{10}$-$s_5$ malfunction, the value of $x_i$ and $a_1$ are interchanged iff $(a_i,
\( a_{i+1} \) is the faulty element. For the malfunction at \( s_3 \) or \( s_{12} \), all \( y_i \)'s have value equal to \( x_i \) and the value of \( a_1 \) is \( x_i \) iff \( (a_i, a_{i+1}) \) is the faulty element. Q.E.D.

Figure 5.8 gives an example. When putting a value at \( a_1 \) for the path \( a_1a_2a_3a_4a_5 \), no output is received at \( a_5 \). All the possible faulty states are depicted above and below the idle elements, assuming all the elements along the path are idle. The established test paths are shown under the assumption that there is no fault. Putting values 0,1,2,3,4 on the inlets, if the outputs are 2,1,2,3,4, then the fault occurs at the binary element \( (a_2, a_3) \) and a malfunction at \( s_{12} \). Active elements and terminals are not shown in the figure for clarity.

5.7. Fault Tolerance for Single Control Faults

In fault diagnosis, detecting faults is essential and locating faults is helpful for manufacturing decisions, repairing replacements, and fault tolerance. Fault tolerance techniques are methods of introducing redundancy into the network such that the network can still meet its specified capability even if there exist faults in the network. When fault tolerance or reliability is discussed, the specified capability and efficiency of the network are important. Many different criteria of fault tolerance of multistage interconnection networks have been proposed in the
Figure 5.8. Locating Single Element Logical Faults along a Path
Since the capability is referred as the permutations a BEMIN can realize, and the efficiency as the time a given permutation can be recognized and physically realized, a BEMIN is said to be fault tolerant if it still can realize the same set of permutations by the same efficiency under certain faults. Apparently, a BEMIN can not tolerate link faults since the links are fully utilized for a permutation involving all inlets and all outlets. The redundancy introduced by a BEMIN lies in the effect of adding stages [Opfe71], [Sowr80]. Other approaches to introduce redundancy are adding additional bypass links and elements, such as using a Clos nonblocking network [Clos53], and enhancing each binary element with fault-tolerant circuits [Step77].

Since a permutation may be realized by a number of settings of a BEMIN, a BEMIN may contain a certain degree of redundancy. We demonstrate this by the following theorems (see Chapter 3 for the definitions of networks).

**Lemma 5.14**: Let an LT alpha and an RT beta be conjugate networks. Then alpha cascaded by beta is rearrangeable even if there is a single control fault $(s_5,s_5)$ or $(s_{10},s_{10})$ in the cascaded network.
Proof: Any transversal network (TN) of size $n$ is rearrangeable and there is a looping algorithm [Opfe71] which can be used to set the network to realize a given arbitrary permutation (see Chapter 3). The looping algorithm can be parallelized to run more efficiently [Nass82]. The setting of the binary elements follows from the recursive construction of the corresponding GBBN.

The algorithm consists of $\log(n)$ levels. In level $i$, it sets the first stage and the last stage binary elements of all $2^{i-1}$ GBBNs from top to bottom to realize the permutations determined by the previous level $i-1$. When setting the binary elements, it chooses a first (last) stage element $b$ arbitrarily and sets its state arbitrarily to be 0 ($s_{10}$) or 1 ($s_{5}$). For the link led into $\text{GBBN}_T$, the corresponding last (first) stage element must be set such that the link labeled the same according to the desired permutation will also be led into $\text{GBBN}_T$. Thus, another link connected to the element just set will be led into $\text{GBBN}_B$, and a first (last, respectively) stage element must be set accordingly.

The process repeats until the link led into $\text{GBBN}_B$ from the starting binary element $b$ is resumed and a "loop" is completed. If all binary elements in the first and the last stages are set, this level is finished; otherwise, choose an arbitrary un-set binary element in the first or last stage to start another loop.
In a transversal network, if a binary element in any stage except the middle stage has a control stuck fault at s_5 or s_{10}, we simply use that binary element as the starting element of a loop when the setting of the network proceeds to that stage in a level. Thus, the network can still be set for any arbitrary permutation.

If the faulty element occurs in the middle stage, it is automatically tolerated since the middle stage of a transversal network is doubled in the network which cascades an LT and an RT.

Q.E.D.

From the lemma above and Theorem 3.18, we have the following.

**Theorem 5.15**: Given an LT alpha and an RT beta of the same size n, there exists a link pattern M such that the network formed by cascading alpha, M, and beta can tolerate any single control faults and realize any permutation of degree n under the looping algorithm.

A half transversal network is one being both an LT and an RT and having the property that the link pattern M in the above theorem is the identity permutation (see Chapter 3). Consider a binary element with a control fault in an HT. If two such HTs are cascaded, a corresponding TN can be obtained by merging the middle two stages. When the number of stages of the HT is even, the two faulty elements in the TN do not appear in the first or the last
stage of the same recursive level. We exemplify this by Figure 5.9, where the size of the network is 16. In the example, the binary element labeled 0 in stage $S_2$ of the HT has a s-a-0 control fault, and by using the looping algorithm, the HT can still realize any arbitrary permutation despite of the existence of this fault. However, we observe that when the number of stages of the HT is odd, some faulty element in the middle stage of the HT will disable it to realize some permutations by two passes under the looping algorithm. Thus, we have the following.

**Theorem 5.16**: Any $m$-stage HT can realize any permutation by two passes. This property preserves even when there is a single control fault in the network, iff $m$ is even.

Therefore, HTs with even number of stages are insensitive to single control faults, while HTs with odd number of stages are not.

### 5.8. Summary

For fault diagnosis of BEMINs, a fault model which contains several levels is given for theoretical analysis and for different realistic situations. The single logical faults are emphasized as the central consideration.

Based on the fault model, the number of tests required are minimized for generalized networks. Single faults can be detected
Figure 5.9. Single Control Fault Tolerance of an HT
by three tests for any network. Single link faults can be detected
and located by three tests for any PTN or PTN\(^{-1}\). For any MFA
network, single line faults can be diagnosed by three tests, single
logical faults can be detected and located by \(3^+\log(\log(n))\) tests
in a presented single logical fault diagnosis procedure, and
multiple control faults can be detected and located by \(2(\log(n))\)
tests.

In asynchronous mode, techniques are used to diagnose any idle
pair of inlets and outlets. A procedure is given to detect and
locate single logical faults, and it can be applied whenever a new
connection is requested. Link faults, control faults, and element
logical faults can all be effectively diagnosed under the single
logical fault model.

Fault tolerance is discussed for BEMINs, and it is found that
the cascading of a pair of conjugate networks can tolerate any
single control faults, preserving the rearrangeability without
losing efficiency.
Chapter 6

CONCLUSION

The concluding remarks are divided into two consecutive parts. In the first part, we summarize major researches conducted in this dissertation. In sequel, possible future research problems and directions are presented, based on the current state of art.

6.1. Contributions

After we presented a framework and a survey in Chapter 1, the efforts of this dissertation have been made into four areas, Chapter 2, 3, 4, and 5, to discuss the feasible range of some design parameters of dynamic interconnection networks with permutation functions. For each chapter one can trace back from the summary section to the proof techniques and results in the content of that chapter; still, we emphasize several points here as an index for further researches.

A dynamic interconnection network consists of a number of switching elements and fixed connections are assumed among the switching elements such that in total the network realizes a set of states. A state of the network is determined by the setting of all
the states of the switching elements in the network. The set of states realizable by the network depends on the possible states of the switching elements and the link patterns connecting the switching elements. If the switching elements are limited to certain types, a designer can change the link patterns to obtain different networks realizing different sets of states. However, the flexibility comes from the link patterns is not obvious in general.

In Chapter 2, by limiting the switching elements to be exclusively binary elements we have shown that the problem of determining whether a given multistage interconnection network with fixed number of stages and hence fixed number of switching elements can realize a given permutation is an NP-complete problem. This result reflects that finding the relation between the link patterns and the set of realizable states of a network in general is close to intuitively what is deemed intractable. Along with the proof, we have observed that the problems of interconnection networks are related to a number of graph theoretically related problems. Thus the algorithmic analysis results found in one field can be applied to the other field. For example, a generalized PPR problem is equivalent to a subproblem of the subgraph homeomorphism problem. As another example, the NP-completeness results related to the vertex cover number and the matching number in a given hypergraph in some very restricted situations have been derived.
By means of different link patterns in connecting a number of stages of small switching elements, it is possible to design networks that are rearrangeable, and less costly, less delay, more flexible, more reliable and with relatively more utilization than a crossbar switch. Given a growing number of different multistage interconnection networks presented in the literature, each with some good properties, few investigations had been done on the general theory of the interconnection structure of these networks. We have provided a theoretical framework to include existing networks and relate them under some subclasses of the class of BEMINs in Chapter 3. In search of some classes of BEMINs with both polynomial-time permutation-recognition and polynomial-time membership-recognition properties, we have presented the new classes of networks called the class of partial transversal networks and their reverse networks, and the class of minimum full access networks and networks with one more stage added to them. The recursive construction of these classes of networks have allowed us to obtain good properties which are shared by all the networks in the same class. Some of these properties have been used in Chapter 5 and we expect them to be used in other situations.

As far as BEMINs are concerned, a circulating network using a BEMIN as the bijection state network functionally simulates a series of repeated pattern BEMINs. Many multistage interconnection network
problems can be transformed to those of circulating networks. The study of circulating networks has been related to group theoretical problems in Chapter 4. The problem of determining if a CN can realize a given permutation and the problem of determining if a CN can realize all permutations are polynomial-time solvable for the case when the CN is specified by its configuration and the case when the CN is specified by the link patterns of the BEMIN which is used as the BSN of the CN. For both problems in both cases, good algorithms found from good characterizations have been presented. The interesting case when the BSN is a BEMIN has been explored to the exact determination of the group generated by the CN. On the other hand, the problem of finding the shortest phase length to realize a given permutation for a given CN even when it is known to be a PCN and the problem of finding the minimum upper bound on the phase length needed to realize any realizable permutation for a CN have been shown to be NP-hard in general when the CN is specified by its configuration.

In order to improve network reliability and availability, it is desirable to have some means of quickly and accurately detecting and locating faults in the network under consideration. In Chapter 5, we have minimized the number of tests needed to detect and locate single logical faults and discussed the fault tolerance property for some classes of BEMINs found and presented in Chapter 2. Fault
diagnosis procedures have been worked out for both synchronous and asynchronous operation mode of any minimum full access network. It has been found that the cascading of a pair of conjugate networks can tolerate any single control faults without losing the efficiency of setting the network to realize an arbitrary permutation.

In all, the techniques and findings presented in this dissertation will aid both theoretical researchers and practical designers in developing future interconnection networks.

6.2. Future Research

There are several related problems and directions which arise as a result of this study and require further investigations.

We have not answered the related question which asks what is the minimum m needed to achieve a rearrangeable n*m BEMIN. The transversal network gives the upper bound to be \(2(\log(n)) - 1\) when \(n = 2^N\). The lower bound of m must satisfy \(2^{mn/2} \geq n!\) by a simple combinatorial reasoning. For \(n = 2^N\) cases, the lower bound is less than the number of stages of a transversal network of the same size when \(N > 2\). It is also open to find the minimum m such that the n*m BEMIN with the same fixed link pattern for each stage has rearrangeability.

We have shown that PPR is polynomially solvable when the number
of inlets and outlets is fixed. However, the algorithm may not be efficient enough in practice when the number of inlets and outlets becomes large. For PPR or VPPR, it is conceivable that there is a certain constant \( c \) such that when the number \( m \) of stages satisfies \( m \geq c \), the problem is \( \text{NP}-\text{complete} \), and when \( m < c \) the problem is polynomially solvable, although we know that \( c \) is small.

Similarly, we do not know whether the class of networks formed by adding two arbitrary stages to a minimum full access network has the property of efficient permutation-recognition. For expanding the efficiently membership-recognizable and permutation-recognizable classes of networks, a related theoretical problem is whether two given BEMINs are topologically equivalent.

The following three subproblems of PPR have not been solved.
The single repeated pattern PPR problem: The link patterns are the same between all stages.
Shuffle PPR: Fix the link patterns to a particular permutation, say the perfect shuffle pattern.
The periodic pattern PPR problem: Given an \( n \times m \) BEMIN, determine if it can realize a given permutation \( P \) by \( k \), \( k > 1 \), passes.

Although some small size networks which are not constructed under the partial transversal network structure are shown to be rearrangeable, whether these networks can be generalized for larger
sizes remains unknown.

A question asking the minimum upper bound phase length for the shuffle-exchange network to realize all permutations is left unanswered. This is actually equivalent to asking for the minimum number of stages a BEMIN using only perfect shuffle pattern to be rearrangeable. A HT can realize any permutation by two passes. But it requires further investigation to the problem whether a LRT, e.g. omega network, or a minimum full access network, can realize all permutations by two passes.

We have found that a CN can be easily tested for capability. In general case, the efficiency optimization poses a hard problem. One important factor we have not paid much attention is the control. That is, the problem of how a permutation is realized by the composition of a sequence of states, each being achieved by setting the state of the bijection state network of the CN. Is it easy for a shift-exchange network to be controlled than for a shuffle-exchange network?

The fault diagnosis techniques presented may be extended to distribute controlled networks, for example, bit-controlled networks. The fault diagnosis and fault tolerance should be investigated for networks beyond the BEMIN structure. For example, the Clos three-stage network may have good fault-tolerant
Although we only limit ourselves to binary elements, the extension to interconnection networks using larger switching elements may be worked out without difficulty in many cases. For example, the concept of minimum full access networks may be expanded to networks of size \( n=3N \) instead of only \( n=2N \). We can also ask similar questions presented in this dissertation for concentration networks, full switches, general connection networks, partitioning networks, and sorting networks. Many techniques presented in this dissertation may similarly be applied to the pertinent problems in these networks.
BIBLIOGRAPHY


**Bene75**  

**Bent78**  

**Berg73**  

**Bern82**  

**Breu76**  

**Cant72**  

**Catt79**  

**Chun79**  

**Clos53**  

**Davi81**  

**Diac82**  

**Dias82**  

**Dugu59**  

**Even76**  


Kuhn81  KUHN, R. H. AND PADUA, D. A. Tutorial on Parallel Processing, IEEE, 1981.


LOVASZ, L. On Two Minimax Theorems in Graph, *J. of Combinatorial Theory (B)* 21, (1976), 96-103.


Sieg79 SIEGEL, H. J. Interconnection Networks for SIMD Machines, Computer, (June 1979), 57-65.


<table>
<thead>
<tr>
<th>Reference</th>
<th>Author(s)</th>
<th>Title and Details</th>
</tr>
</thead>
</table>