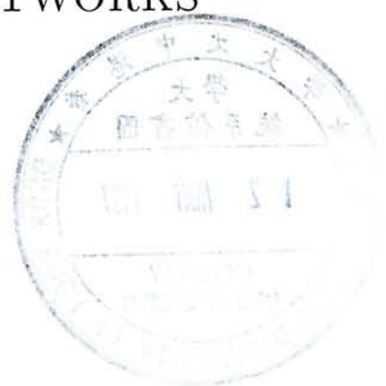
PARALLEL ROUTING ALGORITHMS IN BENES-CLOS NETWORKS



By

Soung-yue Liew

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Information Engineering

The Chinese University of Hong Kong

June 1996

Acknowledgement

I would like to express my sincere gratitude towards my supervisor Prof. Tony T. Lee and my brother Soung C. Liew for their advices, supports and encouragements, which make this 2-year research period a fruitful and rewarding experience.

I would also like to thank Philip P.T. To for his valuable comments on this thesis.

Abstract

A new parallel algorithm for route assignment in Benes-Clos network is studied in this thesis. In packet switching systems, switch fabrics must be able to provide internally conflict-free paths simultaneously and to accommodate packets requesting for connections in real-time as they arrive at the inputs. Most known sequential route assignment algorithms, such as the looping algorithm for Benes networks or Clos networks, are designed for circuit switching systems where switching configuration can be rearranged at relatively low speed. Most existing parallel routing algorithms are not practical for packet switching because they either assume the set of connection requests is a full permutation or fail to deal with output contentions among the set of input packets. In this thesis, we develop a parallel routing algorithm by solving a set of Boolean equations which are derived from the connection requests and the symmetric structure of the Benes network. Our approach can handle both the partial permutations and the output contention problem easily. The time complexity of our algorithm is $O(log^2 N)$, where N is the network size. Furthermore, we extend the algorithm and show that it can be applied to the Clos network if the number of central modules is a power of two.

Keywords — Benes networks, Clos networks, symmetric self-routing, internally conflict-free, Boolean equations, parallel routing algorithms, output contention.

Contents

List of Figures

Chapter 1

Introduction

In the past few decades, computer and communication technologies was growing up rapidly due to the various demands of communications services. Wide-area and local-area computer networks have been extensively deployed to interconnect computers throughout the world. In addition to transmission, switching plays an important role in communications network because it changes the terminals of connections dynamically in order to increase the utilization of shared resources. For example if there are N users in a communication system, instead of using C_2^N independent transmission lines to connect each two of the terminals (i.e. fully connected structure), a switching facility and N transmission lines would be sufficient for any connection request, as shown in Figure 1.1

Classically, the communications network is designed as a circuit-switched network. That is, a circuit or a connection is established between two end users when there is a request, and this circuit will not be shared with other end users at the same time. A connection request is said to be "blocked" if network resources are not available to set up the connection to satisfy the request. The

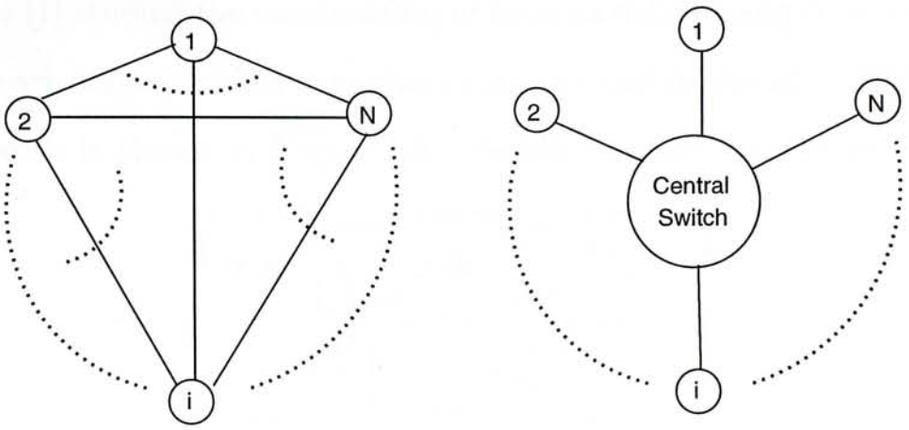


Figure 1.1: Resources sharing through a switch

blocking may occur inside a switch node even if enough capacity is provided via the transmission lines. The challenge of circuit switch design is how to establish non-blocking connections with minimum network complexity.

With reference to Figure 1.2, an $N \times N$ switch can be simply constructed by cross-bar structure to avoid internal blocking. That is at any time, at most one crosspoint of each column or row can be turned on. However if N is large, it is not practical to construct such a huge switch because of high routing complexity. In

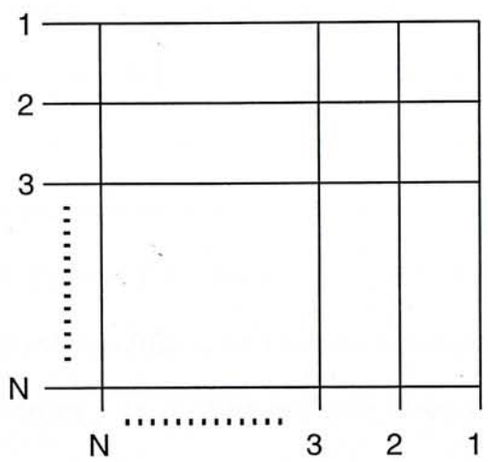


Figure 1.2: An $N \times N$ crossbar switch

1953, Clos [1] studied the construction of large switched using connected stages in which each stage contains a number of smaller switch modules. A three-stage Clos network is shown in Figure 1.3. Switch modules are arranged in three

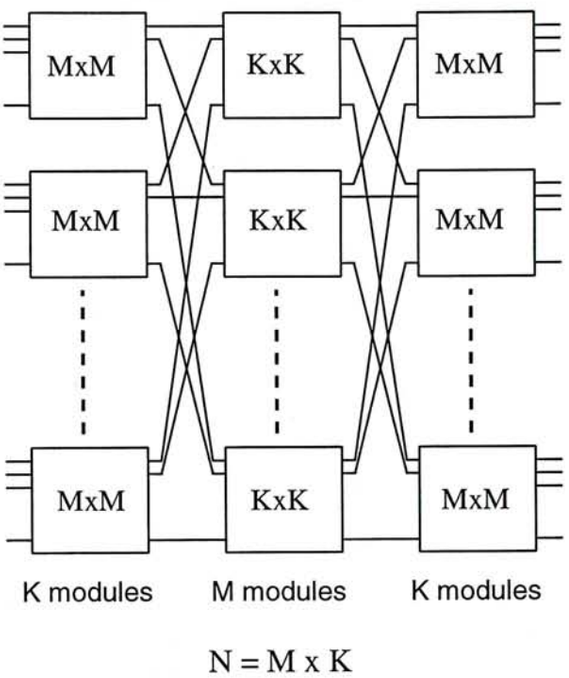
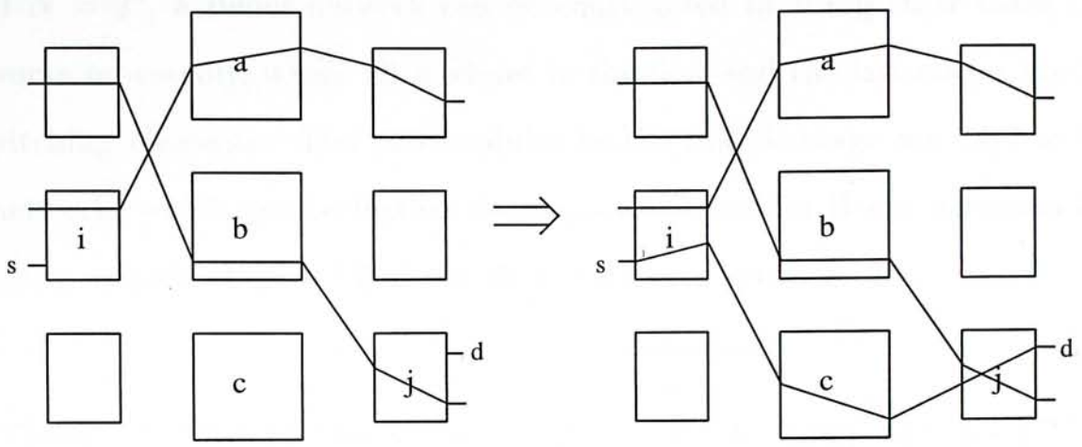


Figure 1.3: Three-stage Clos network

stages and there is a unique interlink connecting any pair of modules in two adjacent stages. Assume that each of these modules is internally non-blocking. The routing problem is to select one of the central modules for each request such that any two selected connections will not share the same internal link. An example is shown in Figure 1.4. To set up a call between s and d, the call cannot be routed to central module a or central module b because there is only one link from i to a and from j to b. The call can then only be routed to central module c.

It is obvious that increasing the number of central modules provides more alternative paths to established a connection for a request. If the number of the



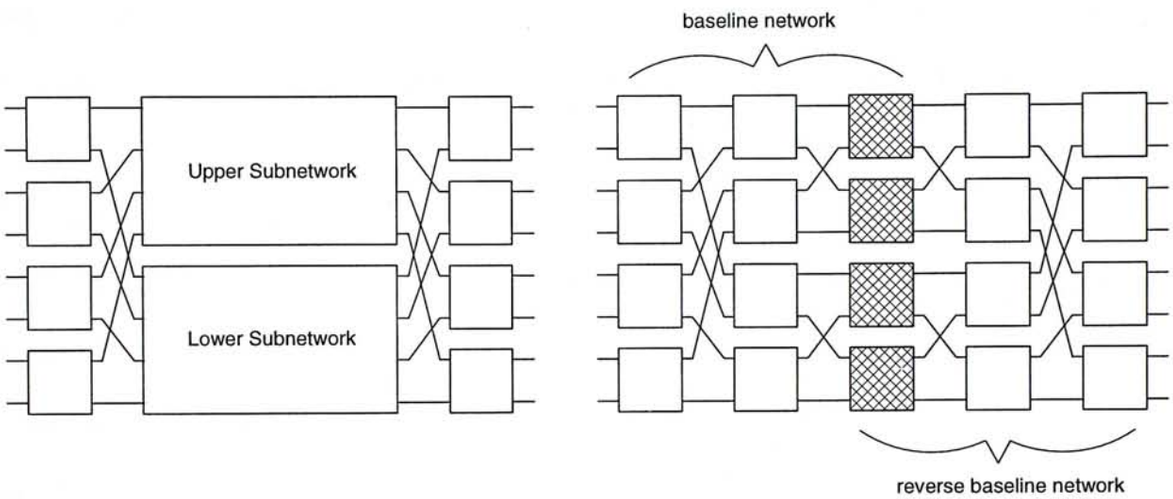
A central module assigned to input s and output d

Figure 1.4: Connection set up in three-stage Clos network

central modules is sufficient to establish a path for a new connection request provided that some existing connections can be rearranged (i.e. re-routed to different central modules), then the network is said to be rearrangeably non-blocking. It has been shown in [2] that if the number of central modules is equal to the number of inputs (outputs) in an input module (output module), then the network is rearrangeable non-blocking. The example shown in Figure 1.3 is a rearrangeably non-blocking Clos network. Each input module or output module is of size $M \times M$. There are M central modules, each is $K \times K$.

It should be noted, however, that the rearrangement of existing connections is not a trivial task if a path cannot established for a new connection request directly, a centralized rearrangement algorithm is required to compute the new routing configuration. One of the classical routing algorithms is called the looping algorithm ([6], [14]). Because it is sequential algorithm, the time complexity of finding a path for a new connection request is of the order of $O(N)$. This will be stated briefly in the next chapter.

If $N = 2^n$, a Benes network can be constructed by using three-stage Clos networks repeatedly, where all modules in the first and the last stages are 2×2 switching elements. The two modules in the middle stage are $N/2 \times N/2$ subnetworks which can be further decomposed to smaller Benes networks in a recursive manner. Figure 1.5 shows an 8×8 Benes network.



(a)Three-stage structure (b)Complete structure

Figure 1.5: An 8×8 Benes network

The Benes network has $2log_2N-1$ stages and each stage contains $N/2$ switching elements. Since a Benes network is a special kind of the Clos network, so the routing in the network can be computed by looping algorithm recursively in log_2N iterations.

The Benes network has received much attention as an interconnection network because of its recursive structure and modularity. Actually the Benes networks have been used in some real parallel computer systems for interprocessor or processor-memory communications.

In circuit switching, a circuit is assigned to the same input-output connection request in the entire length of conversation. Even though there are intermitted

pauses during the conversation, network resources cannot be used by another connection request, leading to a low utilization. As a result, packet switching mechanism, such as Asynchronous Transfer Mode (ATM), is devised to solve this problem.

In packet switching, the successive time slots of a path may contain the packets from different inputs to different outputs. Since the corresponding destination address of the packet input from the same port may change every time slot, so the route assignment must be recomputed in real-time. To avoid packet loss without sacrificing switching speed, the route assignment corresponding to a set of input-output connection requests need to be resolved as quickly as possible. Consequently, the sequential looping algorithm is not applicable in packet switching because its time complexity is relatively high compared to the duration of a time slot. In order to obtain an algorithm with time complexity comparable to the duration of a time slot, it is therefore necessary to consider parallel algorithms.

Parallel algorithms in Benes networks and Clos networks have been investigated for many decades and a number of algorithms have been developed. However most existing parallel routing algorithms are not practical for packet switching, because they either assume the set of connection requests is a full permutation ([4], [9]) or fail to deal with output contentions among the set of input packets ([8], [12]).

In this thesis, we proposed a new parallel algorithm for route assignment in Benes-Clos network. To compare with other algorithms, our algorithm can solve the routing problem naturally without making any unrealistic assumptions. Furthermore, we show that our algorithm can deal with partial permutations and

output contentions easily.

The development of our algorithm is based on the observation that the Benes network, recursively constructed from the Clos network, exhibits a symmetric topological structure. That is, the Benes network can be considered as the cascaded combination of an omega network and a reverse omega network, as shown in Figure 1.5, they are the baseline network and the reverse baseline network. The omega network and the reverse omega both belong to the class of Multi-stage Interconnection Network (MIN). An MIN possesses the unique path self-routing property. That is, each input and output are connected by a unique path and the path from an input to an output can be determined by the destination address. Thus, each input-output path in the Benes network actually consists of two sub-paths — one in the omega network and the other in the reverse omega network. These two sub-paths can be joined at any one of the central modules to form a complete path. Therefore, there are $N/2$ alternative paths for each input-output pair in the Benes network, where N is the network size. The routing problem is to select one of these alternative paths for each request such that any two selected paths will not share the same internal link.

For any complete path in the Benes network, the sub-path from the input to the middle stage and the sub-path from the output to the middle stage must have the same binary "destination address" which is illustrated in Figure 1.6. This is because the input and the output must meet at the same central module in order to establish a complete path. We call this the symmetric routing constraint.

In the case of an $N \times N$ Benes network, it takes $n - 1$ steps to reach a central module from any input or any output. Therefore the binary "destination address" is composed of $n - 1$ bits. Let the binary address of the sub-path from

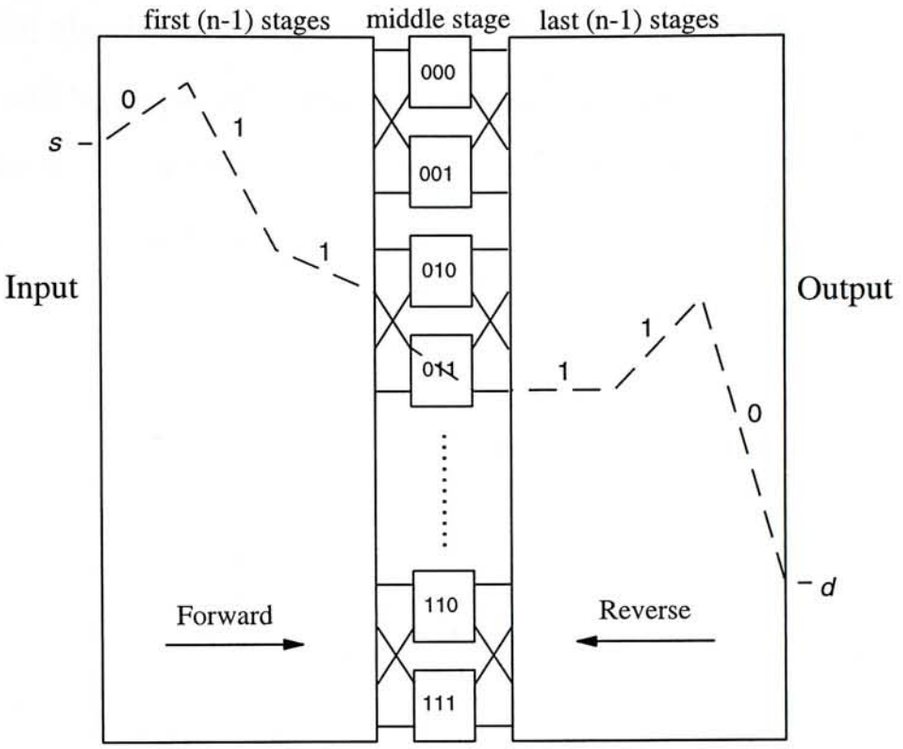


Figure 1.6: Symmetric routing

input to the middle stage be $\alpha_1\alpha_2 \ldots \alpha_{n-1}$ and the binary address of the sub-path from the output to the middle stage be $\beta_1\beta_2 \ldots \beta_{n-1}$. By the symmetric routing constraint, we must have $\alpha_i = \beta_i$ for all i where $1 \leq i \leq n-1$. Since each bit determines the state of a 2×2 switching element along the path, we show that a set of Boolean equations can be established for each stage recursively, with the Boolean variables representing the state of the switch elements involved. The solution of this set of Boolean equations can be calculated by distributed and parallel procedures.

As far as routing is concerned, a Clos network with 2^m central modules is equivalent to a fictitious Benes network. Thus, our algorithm can also be applied to the Clos network in a straightforward manner.

This thesis is organized as follows. Chapter 2 describes the basic principles

of sequential algorithms and parallel algorithms. In Chapter 3, A new parallel algorithm will be investigated after two fundamental constraints of route assignment in Benes-Clos networks have been studied. Comparisons and Conclusions will be given in Chapter 4.

Chapter 2

The Basic Principles of Routing Algorithms

The route assignment problem in a Clos network can be formulated by the edge-coloring problem in a bipartite graph. This formulation was first studied by Lev, Pippenger and Valiant in [3]. With reference to Figure 2.1, given a set of connection requests in a three-stage Clos network, the input modules and the output modules can be considered as the vertices in the two disjoint sets $\mathbf{V_1}$ and $\mathbf{V_2}$, respectively, in a bipartite graph $\mathbf{G}(\mathbf{V_1}, \mathbf{V_2})$. Each central module can be regarded as a particular edge-color. If a connection requests a path from input module i to output module j (an edge is connected from vertex i of $\mathbf{V_1}$ to vertex j of $\mathbf{V_2}$), then a central module should be assigned to this request in such a way that other ports in these two modules will not share the same central module (no adjacent edges have the same color). In other words, the edge-coloring of a bipartite graph corresponds to a particular set of route assignment. In this chapter, we will discuss some important issues of the edge-coloring problem in

a bipartite graph, which is the foundation of the construction of our algorithm.

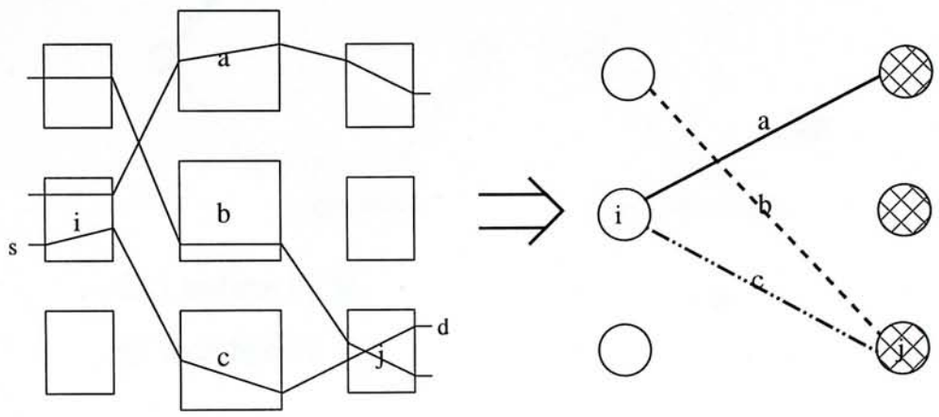


Figure 2.1: Bipartite graph corresponding to three-stage Clos network

2.1 The principles of sequential algorithms

Many algorithms was proposed to solve the edge-coloring problem in a bipartite graph. The sequential algorithms, which make use of single processor, will be discussed in this section. We will first solve the edge-coloring problem in the bipartite graph with maximum degree two.

2.1.1 Edge-coloring of bipartite graph with maximum degree two

In a bipartite graph with maximum degree two, each vertex is connected by at most two edges. For any two adjacent vertices, one is from $\mathbf{V_1}$ and another is from $\mathbf{V_2}$. Each component in such a bipartite graph can be a cycle chain or a non-cycle chain as shown in Figure 2.2. If a component is a cycle chain, it must contain even number of vertices and edges, because **odd cycle chain**

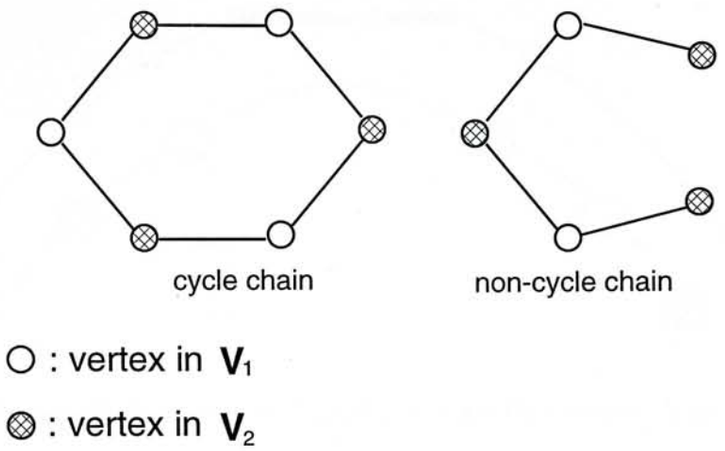


Figure 2.2: A bipartite graph with maximum degree two

can never exist in a bipartite graph. It follows from König's Theorem [16], therefore, a bipartite graph with maximum degree two can be edge-colored with only two distinct colors. After edge-coloring such graph with two colors, the color of an edge must be same to the color of the edge which is $2k$ vertices away, and must be distinct to the color of the edge which is $2k-1$ vertices away. As a result, we have the following theorem.

Theorem 1 *For any two end-vertices which are from* $\mathbf{V_1}$ *and* $\mathbf{V_2}$ *respectively, if their edges are colored distinctly, then they are disconnected.*

Proof : With reference to Figure 2.3, if end-vertex i from $\mathbf{V_1}$ and end-vertex j from $\mathbf{V_2}$ are in the same chain (i.e. if they are connected), then the chain must consist of even number of vertices, so the edges incident to i and j respectively must be colored identically. Therefore, if the edges incident to i and j respectively are colored distinctly, as shown in Figure 2.4, they are disconnected.

□

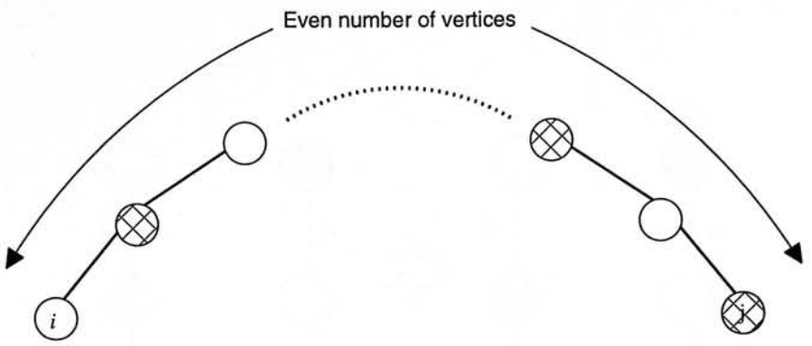


Figure 2.3: Assume i and j are in the same chain

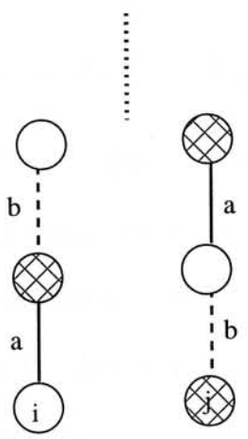


Figure 2.4: i and j are disconnected

We will show how to edge-color a bipartite graph with maximum degree two by using color a and color b as the following.

Initially, we can color an edge arbitrarily.

Suppose now we want to color an uncolored edge which joins vertex i to vertex j, there are two cases to be considered:

Case I: With reference to Figure 2.5, if there exists a color that has **not** been used to color both edges incident to vertex i and vertex j respectively before coloring the uncolored edge, then the edge can be colored directly by this missing color.

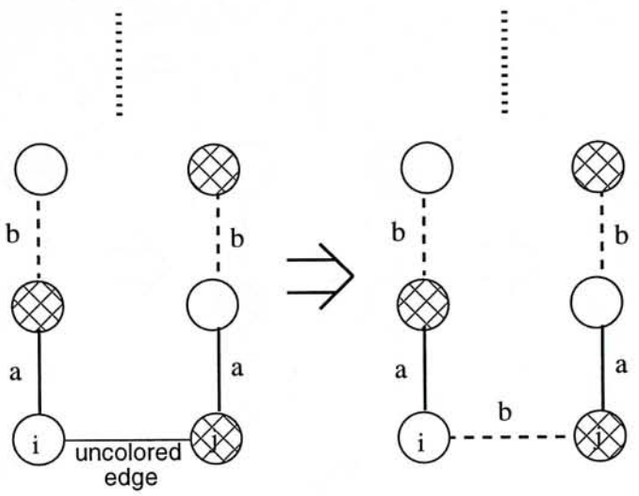


Figure 2.5: Color b has not been used at i and j before the new edge is colored.

Case II: With reference to Figure 2.6, if both colors have been used, for example the edge incident to vertex i has been colored by a, and the edge incident to vertex j has been colored by b. Consider only those **edges that have been colored**, according to theorem 1, vertex i and vertex j must be in different **colored chains**, so, we can simply choose one of the colored chains and interchange the two colors throughout the chain, then we can use the method described in Case I to color the edge.

This strategy is very basic and important because it can be extended to edge-color the bipartite graph with maximum degree M, where M is an integer which is greater than one.

2.1.2 Edge-coloring of bipartite graph with maximum degree M

At the process of edge-coloring a bipartite graph with maximum degree M, let S_v be the set of the colors that have been used to color the edges incident to

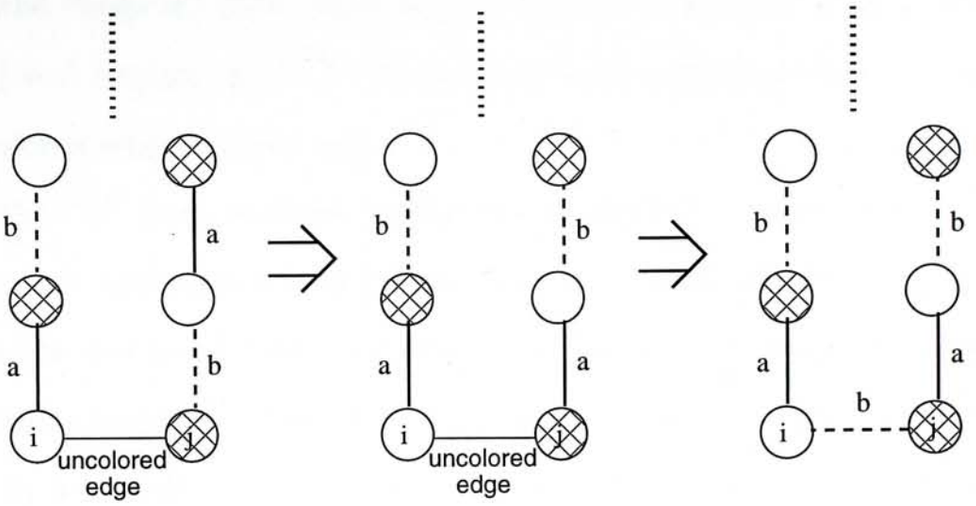
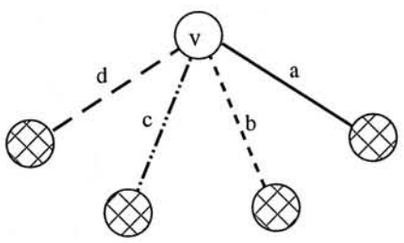


Figure 2.6: The new edge can be colored after rearrangement

vertex v. An example is shown in Figure 2.7. Note that $|S_v| \leq M$.



$S_v = \{a, b, c, d\}$

Figure 2.7: The set of colored edges incident to vertex v

Suppose an uncolored edge which joins vertex i to vertex j is going to be colored now. If there are only L distinct colors available, and if $|S_i \cup S_j| < L$, then the uncolored edge can be colored directly by one of the missing colors at both i and j.

Theorem 2 *If $|S_i \cup S_j| = L$, an uncolored edge can be colored by rearranging the existing colors if and only if $|S_i - S_j| \geq 1$ and $|S_j - S_i| \geq 1$.*

Proof :

For the "only if" part, suppose $|S_i - S_j| = 0$, then $S_i \subset S_j$. Therefore, $|S_i \cup S_j| = L$ implies $|S_j| = L$. Thus, there is no remaining color to edge-color the uncolored edge incident to j.

For the "if" part, without lossing the generality, suppose color a is in S_i but not in S_j and color b is in S_j but not in S_i. With reference to Figure 2.8, consider the sub-graph which contains only those edges colored by a and b, it is a bipartite graph with maximum degree two. Therefore, the new edge can be colored by a or b after rearranging some edge-colors by the method described in Case II. □

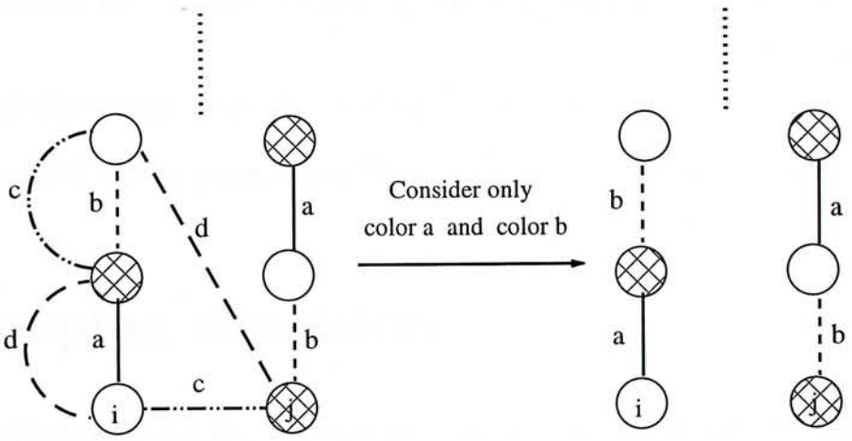


Figure 2.8: Sub-graph contains only edges colored by a and b

Theorem 3 *If the maximum vertex-degree of a bipartite graph is M, then the bipartite graph is L-edge-colorable if and only if $L \geq M$.*

Proof :

For the "only if" part, it is obvious that a vertex with degree M need at least M distinct colors for edge-coloring.

For the "if" part, we will prove that a new edge can always be colored suppose $L \geq M$ colors are provided.

If there is an uncolored edge joining vertex i and vertex j, then $|S_i| \leq M-1$ and $|S_j| \leq M-1$.

For $|S_i \cup S_j| < L$, the edge-coloring is trivial.

If $|S_i \cup S_j| = L$,

$$|S_i - S_j| = |S_i \cup S_j| - |S_j| \geq L - (M-1) \geq M - M + 1 = 1$$

Similarly, $|S_j - S_i| \geq 1$.

According to theorem 2, there exists a rearrangement for edge-coloring. □

Using the theorems and the strategy described above, a sequential routing algorithm, called the looping algorithm, will be introduced in the next section.

2.2 Looping algorithm

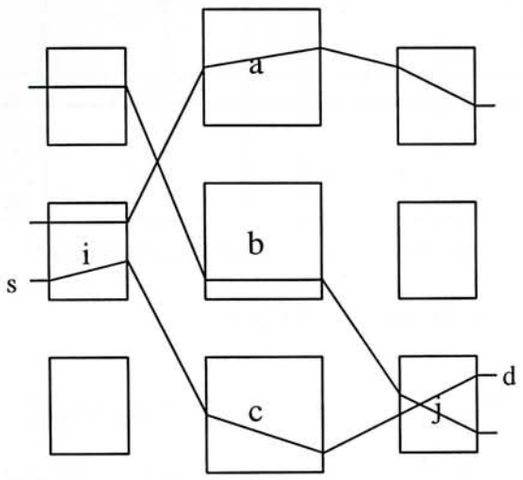
We shall first introduce a matrix notation devised by Paull for representing paths in the Clos network [14].

2.2.1 Paull's Matrix

The Paull's Matrix is a $K \times K$ matrix in which each row represents an input module and each column represents an output module. If input modules i and output module j are connected via central module c, then we enter the symbol c into the entry (i, j) of the matrix. This is illustrated in Figure 2.9.

There is a restriction of this matrix, the symbols in each row or column must be distinct. Since each input module (output module) contains M input ports

(output ports), so each row (column) may contain at most M symbols.

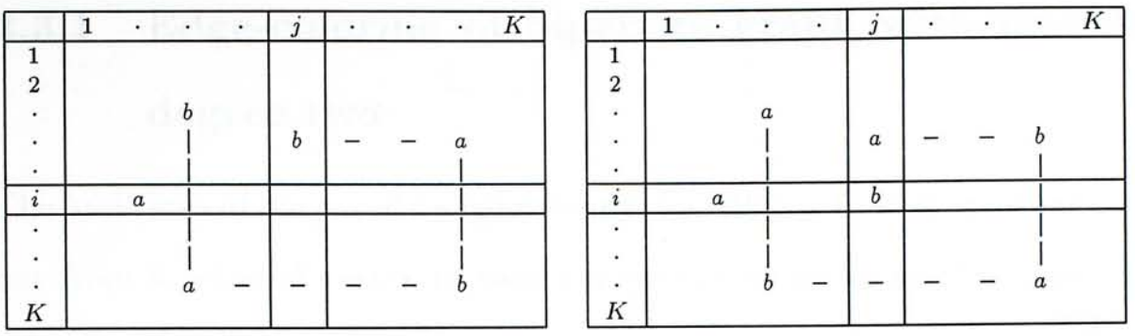


	1	2	···	j	···	K
1						
2						
⋮				b		
i			a	c		
⋮						
K						

Figure 2.9: Paull's Matrix for representation of connections

2.2.2 Chain to be rearranged in Paull's Matrix

Suppose now we want to establish a new connection between the input module i and the output module j, if there is at least a symbol excluded from both row i and column j, the new connection can be set up directly. However if there is not, rearrangement will be needed. Suppose that all symbols except b occur in row i and all symbols except a occur in column j. The motivation of the looping algorithm is to change the symbol b in column j to symbol a so that b can be put into entry (i, j) (or reversely). As shown in Figure 2.10, if we change the symbol b in column j to symbol a, there might be a symbol a already in the row occupied by the b in column j, to avoid contradiction, we must change this existing a to symbol b. However there could be another contradiction in the next column, so the procedure will be executed again and recursively until a row or a column not containing both b and a is reached.



Searching paths in Paull's matrix Connection set up after rearrangement

Figure 2.10: Rearrangement in Paull's Matrix

Note that the basic principle of this algorithm is similar to adding a colored edge in a colored bipartite graph with maximum degree M, it follows from the Theorem 3 derived in the previous section, there exists a rearrangement such that any new connection request will obtain a path.

To analyze the time complexity of looping algorithm with single processor, suppose that time complexity of a row or a column search is of $O(1)$, the number of the steps needed for a rearrangement is equal to the total number of rows and columns that have been searched before the searching halts. As there are K columns and K rows, so the maximum number of steps needed is $2K - 1$, in other words, it is $O(N)$.

2.3 The principles of parallel algorithms

Again, we will discuss first the principle of edge-coloring bipartite graph with maximum degree two.

2.3.1 Edge-coloring of bipartite graph with maximum degree two

The basic idea of the parallel edge-coloring algorithm is to propagate the coloring out from a selected vertex in each component at an exponential rate. Such selected vertex is always said to be the *representative* in the component. Suppose all vertices are numbered before edge-coloring, we will select the vertex with minimum index in each component as the representative of that component.

We shall introduce the searching procedure which plays the main role in the parallel algorithm. Let $A^k[i]$ be the vertex reached by i after k steps of searching, the searching procedure can be defined recursively by

$$A^k[i] = A^{k-1}[A^{k-1}[i]]$$

for $0 \leq i, A^k[i] \leq N-1$ and $k \geq 1$.

Note that $A^0[i]$, for all i, must be given initially.

An example is shown in Figure 2.11. On a searching branch of vertex i,

$A^0[i] = j,\ A^0[j] = k,\ A^0[k] = l,\ A^0[l] = x,\ \cdots$.

After one step of searching,

$A^1[i] = A^0[A^0[i]] = A^0[j] = k,\ A^1[k] = A^0[A^0[k]] = A^0[l] = x,\ \cdots$.

After two steps of searching,

$A^2[i] = A^1[A^1[i]] = A^1[k] = x,\ \cdots$.

And so on.

The motivation of the searching procedure is to determine the components and the representative of each component, because it performs a type of transitive closure.

If there are total N vertices in this bipartite graph, we will show that vertex

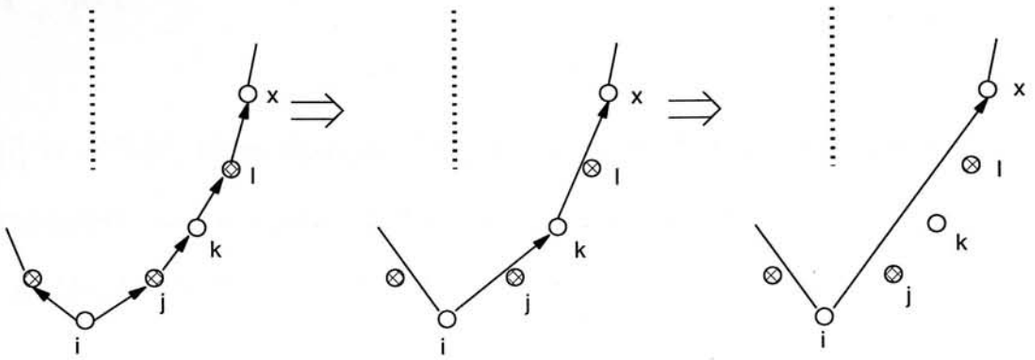


Figure 2.11: Propagation of searching from vertex i

i, starts from its adjacent vertex, takes at most $\lceil log_2(N-1) \rceil$ steps to search through the component to which it belongs. Let $dist\{i,j\}$ be the number of edges from vertex i to vertex j (i.e. distance of i and j), and $EN[i]$ be the end-vertex of the searching from i. According to the searching principle, we have

Theorem 4 *Node i will point to the node $EN[i]$ after $\lceil log_2\ dist\{i, EN[i]\} \rceil \leq log_2(N-1)$ steps of searching.*

Proof : First, we have to prove $dist\{i, A^k[i]\} = 2^k$ if $A^k[i]$ is not $EN[i]$, note that $EN[A^k[i]] = EN[i]$.

Initially, it is obvious that

$dist\{i, A^0[i]\} = 1 = 2^0$.

Assume it is true that

$dist\{i, A^{k-1}[i]\} = 2^{k-1}$, for $k \geq 1$

We have

$dist\{i, A^k[i]\}$

$= dist\{i, A^{k-1}[i]\} + dist\{A^{k-1}[i], A^k[i]\}$

$= dist\{i, A^{k-1}[i]\} + dist\{A^{k-1}[i], A^{k-1}[A^{k-1}[i]]\}$

$= 2^{k-1} + 2^{k-1}$

$= 2^k$.

If $A^k[i]$ is $EN[i]$, then $dist\{A^{k-1}[i], A^k[i]\} \leq 2^{k-1}$, however, the searching will be completed once i reaches $EN[i]$, so the number of searching steps needed is $\lceil log_2\ dist\{i, EN[i]\}\rceil \leq log_2 N - 1$ □

If a vertex is connected by two edges, it will search along two branches.

With searching procedure, after k steps of searching vertex i can find the vertex with minimum index within distance 2^k from i. After $\lceil log_2(N-1)\rceil$ steps of searching, the representative of each component as well as the distance from each vertex to the representative can be determined. When the procedure is completed, the two edges incident to the representative can be colored by two distinct colors arbitrarily, then, other edges in the component can be colored immediately. The further discussions will be given in the next chapter.

2.3.2 Edge-coloring of bipartite graph with maximum degree 2^m

Consider the bipartite graph with maximum degree M, if $M = 2^m$, we can split each vertex into $M/2$ sub-vertices in such a way that each sub-vertex is with maximum degree two, as shown in Figure 2.12.

To color the edges incident to these sub-vertices, by color a and color b for example, we can apply the parallel algorithm described in the previous section. After one iteration of edge-coloring, as shown in Figure 2.13, each original vertex contains one set of edges colored by a and another set of edges colored by b, where each set may consist of at most $M/2$ edges.

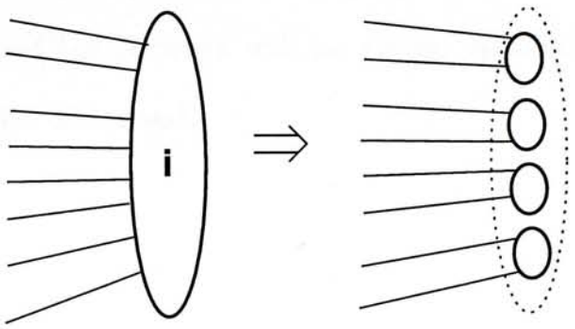


Figure 2.12: To split a vertex into $M/2$ sub-vertices

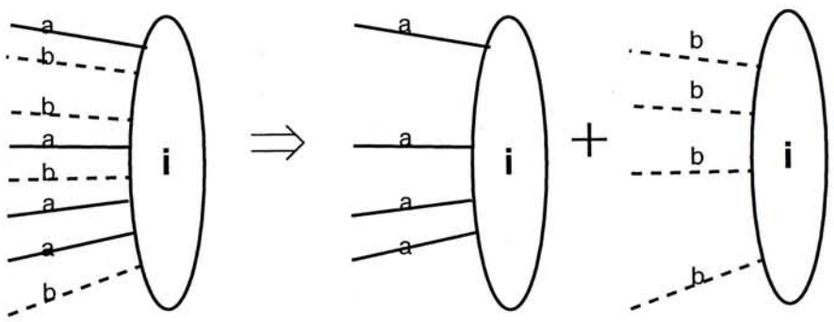


Figure 2.13: Two independent sets of a vertex

If we consider only those edges colored by a (or only those colored by b), a bipartite sub-graph with maximum degree $M/2$ can be obtained from the original graph. This sub-graph can be divided again into two sets such that each contains a distinct color by using the same strategy described above. After M iteration of splitting and partial edge-coloring, the original bipartite graph will be edge-colored by M distinct colors.

If M is not a power of two, then the above strategy can edge-color such bipartite graph if $L = 2^{\lceil log_2 M \rceil}$ colors are provided, however, it is not an optimized algorithm because the minimum number of colors needed is $M < L$. There is still no full parallel and optimized algorithm for M not a power of two.

In the next chapter, a new parallel routing algorithm based on the fundamental properties of Benes-Clos network will be given. Also some implementations of the algorithm will be discussed.

Chapter 3

Parallel routing algorithm in Benes-Clos networks

The topological structure of Benes-Clos networks exhibits two fundamental properties: (1) Symmetry; (2) Unique interlink connecting two modules in the adjacent stages. These properties give constraints for route assignment in the networks. In this chapter, the parallel routing algorithm in Benes networks will be studied first, and then some of the applications will be introduced.

3.1 Routing properties of Benes networks

In this section, the routing problem in a Benes network is formulated as a set of *Boolean equations*. By transitive closure, a solution can be found within time $O(logN)$ and space $O(N)$. The algorithm will be given in the next section.

3.1.1 Three-stage structure and routing constraints

The Benes network $B(n)$ with $N = 2^n$ inputs and N outputs can be considered as a three-stage network as shown in Figure 3.1, where all $N/2$ switch modules in the first stage and $N/2$ switch modules in the last stage are 2×2 switching elements. The two switch modules in the middle stage are $N/2 \times N/2$ subnetworks and each can be constructed by a $B(n-1)$ in a recursive manner. Each 2×2 switching element has a unique link (link with label 0) connecting to the upper subnetwork $B_0(n-1)$; and a unique link (link with label 1) connecting to the lower subnetwork $B_1(n-1)$. If a packet at an input (output) switching element is sent to (received from) the link labeled by α (β) , where $\alpha, \beta \in \{0,\ 1\}$, then α (β) is called the *forward routing bit* FRB (*reverse routing bit* RRB) of that packet.

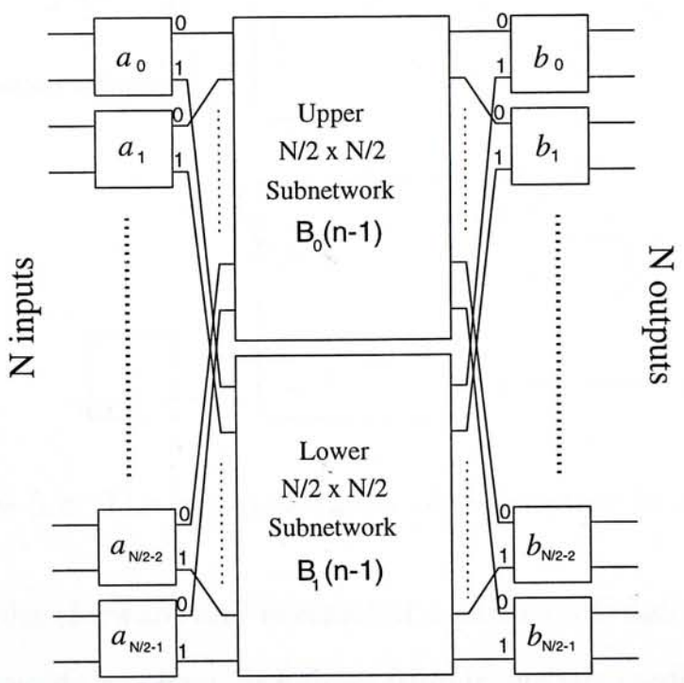


Figure 3.1: The three-stage structure of the Benes network $B(n)$

With reference to Figure 3.2, the routing in a $B(n)$ is *non-blocking* if and only if the following is satisfied.

1. *Symmetric routing constraint* :

   The FRB of an input s and the RRB of an output d must be the same if s and d are in a connection request. It is because they must connect to the same central module to establish a path.

2. *Internally conflict-free constraint* :

   Because of the unique internal link property, the FRBs (RRBs) of the two inputs (outputs) in the same input (output) module must be distinct.

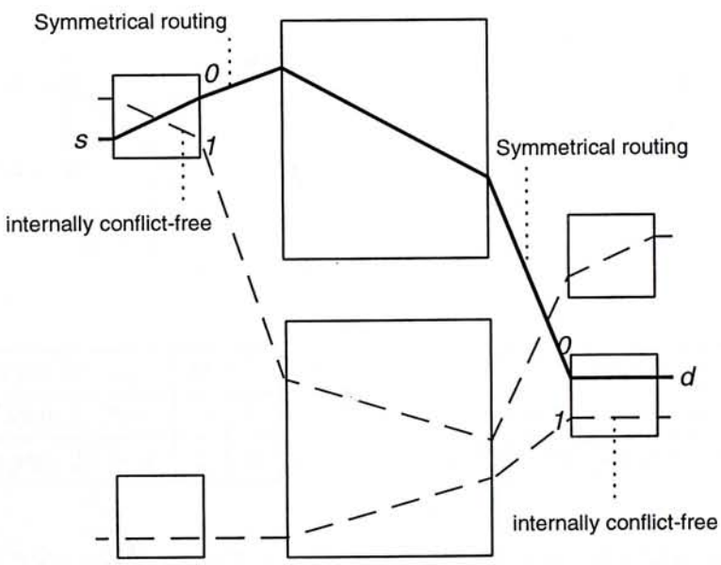
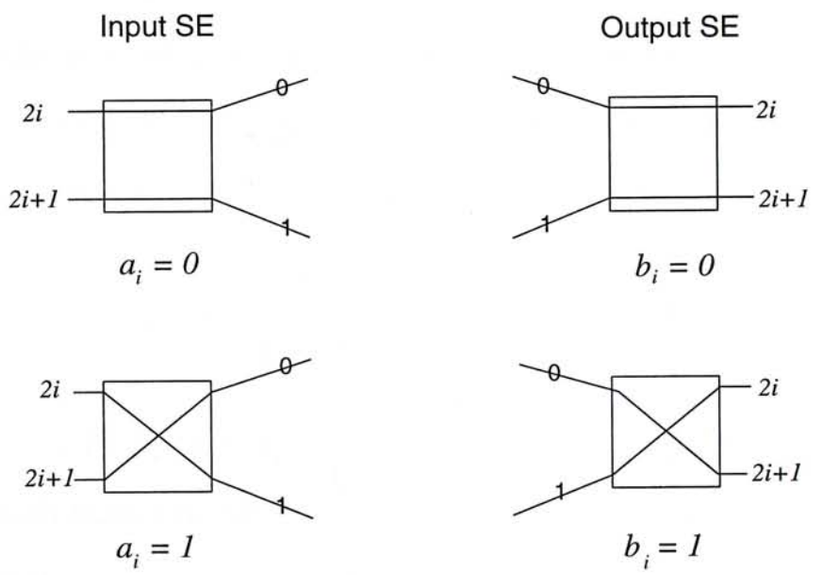


Figure 3.2: The two constraints of the routing in a $B(n)$

The routing bits (forward and reverse) of a packet will determine the state of the switching elements involved. It follows from internally conflict-free constraint that, a switching element in a $B(n)$ can only be in *bar state (state 0)* in which the positions of the two packets remain, or, in *cross state (state 1)* in which

the positions of the two packets exchange. Because of this two-state property, we can use the Boolean variable a_i (b_i) to indicate the state of the ith input (output) switching element. Figure 3.3 illustrates the correspondence between the routing bits and the states of an switching element, where the FRB (RRB) of input (output) $2i$ can be represented by a_i (b_i) and of input (output) $2i+1$ can be represented by $\overline{a_i}$ $(\overline{b_i})$.



state variable a_i	0	1
FRB of input $2i$	0	1
FRB of input $2i+1$	1	0

state variable b_i	0	1
RRB of output $2i$	0	1
RRB of output $2i+1$	1	0

Figure 3.3: The correspondence between the routing bits and the states

It is known that B(n) is a *rearrangeably non-blocking* network [2]. That is, given any set of connection requests in a B(n) without output conflict, there exists a solution of non-blocking route assignment. Such solution motivates our algorithm.

3.1.2 Algebraic interpretation of connection set up problem

Let I and O be the set of inputs and outputs respectively, $I = O = \{0, 1, \cdots, N-1\}$. Let $\pi : I \longrightarrow O$ be an input-output permutation indicating connection requests. If an input is idle, the corresponding output will be represented by x , where x denotes an arbitrary output. An example is given below, the upper row is the ordered inputs from 0 to N-1, and the lower row is the outputs requested by the corresponding inputs. In this example, input 3 is idle, and $\pi(0) = 4, \pi(1) = 5, \pi(2) = 1, \cdots$, etc.

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 1 & x & 0 & 2 & 7 & 3 \end{pmatrix}$$

Let $\alpha : I \longrightarrow \{0,1\}$ and $\beta : O \longrightarrow \{0,1\}$, where $\alpha(k)$ is the routing bit of the forward path from kth input, and, $\beta(k)$ is the routing bit of the reverse path from kth output.

The symmetric routing constraint requires that

$$\alpha(k) = \beta(\pi(k)),$$
$$for\ all\ k \qquad (1)$$

The internal conflict-free constraint requires that

$$\alpha(k) = \overline{\alpha}(k+1)\ ,\ and,\ \beta(k) = \overline{\beta}(k+1),$$
$$for\ k = 0, 2, \cdots, N-2 \qquad (2)$$

The combination of (1) and (2) gives

$$\beta(\pi(k)) = \alpha(k) = \overline{\alpha}(k+1) = \overline{\beta}(\pi(k+1)),$$
$$for\ k = 0, 2, \cdots, N-2 \qquad (3)$$

From the definition of a_i and b_i, which represent the states of ith input switching element and ith output switching element, respectively, the routing bits can be given by

$$\alpha(k) = \begin{cases} a_{\frac{k}{2}} & , \ k\ is\ even \\ \overline{a_{\frac{k-1}{2}}} & , \ k\ is\ odd \end{cases}$$

$$\beta(k) = \begin{cases} b_{\frac{k}{2}} & , \ k\ is\ even \\ \overline{b_{\frac{k-1}{2}}} & , \ k\ is\ odd \end{cases}$$

for $k = 0, 1, \cdots, N-1$

According to this algebraic interpretation, we can set up the Boolean equations for the route assignment in a Benes network. For example, the route assignment of the π given above can be formulated as the following

$$\begin{pmatrix} \alpha(i) \\ \| \\ \beta(\pi(i)) \end{pmatrix} = \begin{pmatrix} a_0 & \overline{a_0} & a_1 & \overline{a_1} & a_2 & \overline{a_2} & a_3 & \overline{a_3} \\ \| & \| & \| & \| & \| & \| & \| & \| \\ b_2 & \neq \overline{b_2} & \overline{b_0} & \neq X & b_0 & \neq b_1 & \overline{b_3} & \neq \overline{b_1} \end{pmatrix}$$

where the equalities are from symmetric routing constraint and the inequalities are from internally conflict-free constraint. X denotes the *don't care term.* Each don't care term, representing an idle input, allows more freedom of routing. To simplify the problem, we can eliminate all the a's, and obtain a set of equations

$$b_2 = b_2\ ,\quad b_0 = X\ ,\quad \overline{b_0} = b_1\ ,\quad b_3 = \overline{b_1}$$

which are called *initializing equations*, because they are the initial conditions

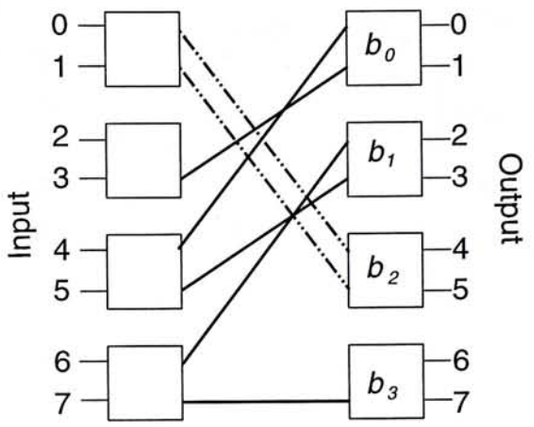
of our algorithms. We can either choose a set of equations which contain only variables $a's$, because these two types of variables are dual.

3.1.3 Equivalent classes

The purpose of our routing algorithm is to solve the initializing equations. Given a connection request π, at most $N/2$ initializing equations can be derived. Let $C = \{b_0, b_1, b_2, \cdots, b_{N/2-1}\}$ be the set of Boolean variables representing the state of the output switching elements. From the relationships given by the initializing equations, C can be partitioned into several equivalent classes such that any $b_i, b_j \in C$, b_i and b_j are in the same class if and only if either one of $b_i = b_j$ or $b_i \neq b_j$ can be true for the non-blocking routing, in other words, they are correlated with π. The set of equivalent classes can be denoted by the partition $C/\pi = \{C_k \mid \forall b_i, b_j \in C_k \text{ iff } b_i \text{ and } b_j \text{ are correlated}\}$ and $C = \bigcup_k C_k$, $C_u \cap C_v = \phi$ for $u \neq v$.

The solution of the initializing equations is not unique. However in each class, if one of the Boolean variables is assigned a value, the value of the remaining variables will be determined, and this independent variable is called the *representative* of the class. We will choose the Boolean variable with the minimum index in a class as the representative of that class. For example in Figure 3.4, b_0 is the representative of the class $\{b_0, b_1, b_3\}$ and b_2 the representative of the class $\{b_2\}$. The motivation of the algorithm can be further interpreted as to determine the independent classes by searching for the representative of each class.

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 1 & x & 0 & 2 & 7 & 3 \end{pmatrix}$$



$\{b_0, b_1, b_3\}$ and $\{b_2\}$ are two independent classes

Figure 3.4: Equivalent classes determined by a π given

3.2 Parallel routing algorithm

3.2.1 Basic principles

To search for the representatives, each initializing equation can be established by a *pointer*, in which we let the Boolean variable with larger index point to the one with smaller index. To distinguish the relationship shown in the equation, two types of pointers are needed. They are, pointer of *type 0* which denotes the two variables are equal, and, pointer of *type 1* which denotes the two variables are counter. We will apply the searching procedure introduced in **Subsection 2.4.1** to determine the independent classes and each representative of the class. Figure 3.5 gives an example to illustrate the initialization and a step of searching. Note that if a node does not point to other node initially, it can be regarded as to point to itself with pointer of type 0.

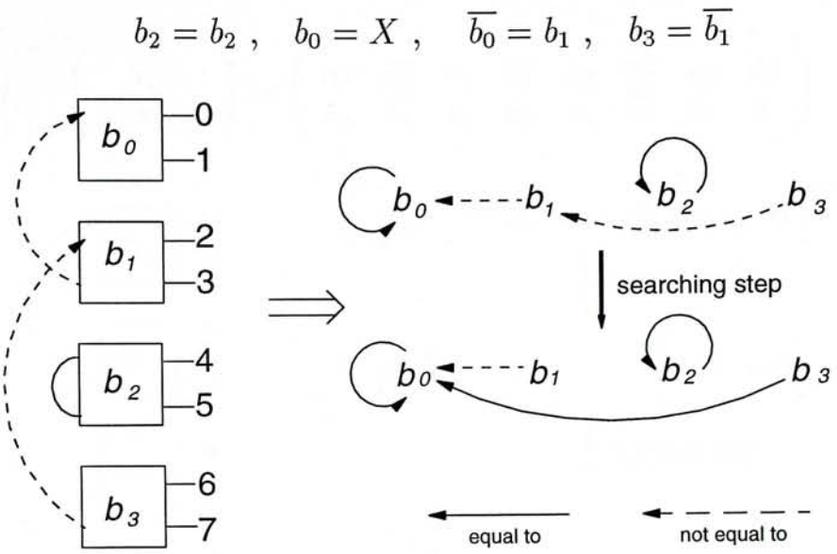


Figure 3.5: Parallel searching of nodes

Since the searching procedure performs a type of transitive closure. As a result, if b_i and b_j are in different classes, then b_i and b_j will never reach each other by the above procedure.

It is possible that a node may initially point to *two* other nodes, such as the b_3 of the example shown in Figure 3.6. In such case, the node will search along both branches, and after several steps of searching it will meet two different ending nodes, i.e. b_0 and b_2 in the example. It is clear that b_2 must be dependent on b_0, since b_2 does not point to any other node, however, this dependency cannot be determined by the above searching procedure. The searching algorithm will be modified by incorporating a merging step to resolve the deadlock. In each step of merging, we compare the deadlock nodes on the two searching ends respectively and let the node with larger index point to the other one with smaller index (b_2 points to b_0 in the example). To avoid confusion, the deadlock resolution will occur in a step of merging only if both ending nodes have been reached.

Notice that also two independent nodes will not reach each other by merging

$$If \begin{pmatrix} \alpha(i) \\ \beta(\pi(i)) \end{pmatrix} = \begin{pmatrix} a_0 & \overline{a_0} & a_1 & \overline{a_1} & a_2 & \overline{a_2} & a_3 & \overline{a_3} \\ b_0 & \overline{b_1} & b_1 & b_3 & b_2 & \overline{b_3} & \overline{b_2} & X \end{pmatrix}$$

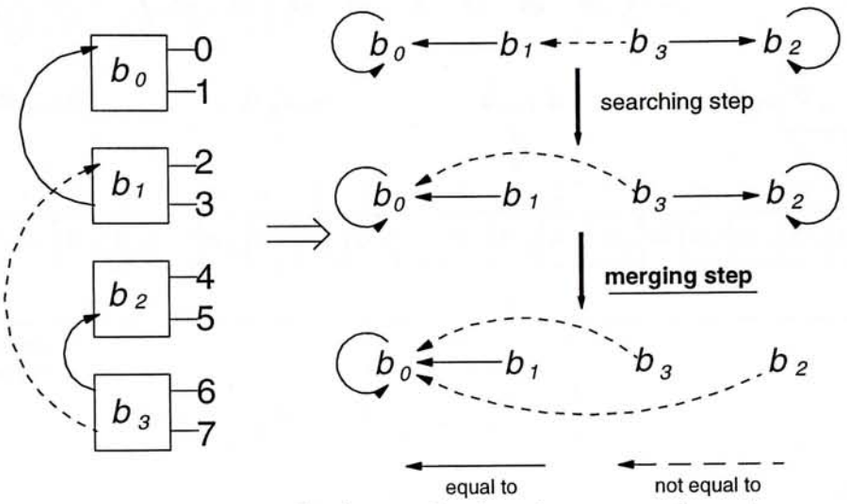


Figure 3.6: Deadlock resolution in a merging step

steps because the merging branches starts from the same node. As a result, searching and merging procedures can determine the independent classes and the representative of each class naturally.

After the representative of each Boolean variable has been determined, a value will be assigned to the representative of each class, and then the states of all *output* switching elements will be determined. Furthermore, the states of all *input* switching elements can be determined by the relations given in $\binom{\alpha(i)}{\beta(\pi(i))}$. These will be shown by an example later.

3.2.2 Initialization

Each pointer described in the previous subsection can be established by a linked list. If there is no output conflict, each node will point to at most two different nodes initially, so only two pointers will be needed for each node. An example of collecting the pointers from the initializing equations and combining them into

two rows of linked lists is shown in Figure 3.7.

$$\begin{pmatrix} a_0 & \overline{a_0} & a_1 & \overline{a_1} & a_2 & \overline{a_2} & a_3 & \overline{a_3} \\ b_2 & \overline{b_2} & \overline{b_0} & b_3 & X & b_1 & \overline{b_3} & \overline{b_1} \end{pmatrix}$$

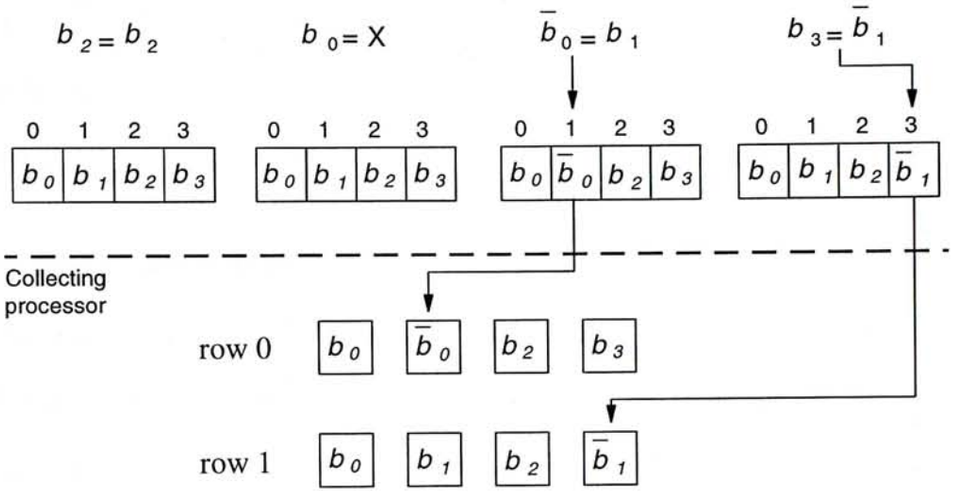


Figure 3.7: Collecting the pointers into two rows of linked lists

Let $A[p,i]$ be the index of the variable in row p at position i, for $p = 0, 1$, $0 \leq i \leq \frac{N}{2} - 1$ and $0 \leq A[p,i] \leq i$. Since the value of each variable is not yet determined, $S[p,i]$, another Boolean variable, will be used to indicate the type of pointer from b_i to $b_{A[p,i]}$, where

$$S[p,i] = \begin{cases} 0 \ , & b_i = b_{A[p,i]} \\ 1 \ , & b_i \neq b_{A[p,i]} \end{cases}$$

$S[p,i]$ is called the **relative state bit** and can be determined by the following equation

$$S[p,i] = b_i \oplus b_{A[p,i]}$$

where "$\oplus$" is the "exclusive or" operator.

3.2.3 Algorithm

Main algorithm

For $k \longleftarrow 1$ *to* m *(m is O($log_2 N$))*

For $0 \leq i \leq \frac{N}{2} - 1$ *(Parallel steps)*

call Searching

call Merging

(1) Searching procedure

In this procedure, the representative of each node as well as the relative state bit will be searched and determined.

For $0 \leq p \leq 1$

$S[p,i] \longleftarrow S[p,i] \oplus S[p, A[p,i]]$;

$A[p,i] \longleftarrow A[p, A[p,i]]$;

(2) Merging procedure

In this procedure, the relation of two different ending nodes will be determined once they have been reached by the same node. Let Y_i be the number of nodes pointed by node i, that is $Y_i \in \{0,1,2\}$. The merging procedure will be further modified if node i points to only one node, i.e. if $Y_i = 1$, we let the two linked lists in column i be the same.

If Y_i=2 and $Y_{A[0,i]}$=0 and $Y_{A[1,i]}$=0

then if $A[0,i] > A[1,i]$ then

$A[0, A[0,i]] \longleftarrow A[1,i]$;

$A[1, A[0,i]] \longleftarrow A[1,i]$;

$S[0, A[0,i]] \longleftarrow S[0,i] \oplus S[1,i]$;

$S[1, A[0,i]] \longleftarrow S[0,i] \oplus S[1,i]$;

$Y_i \longleftarrow 1;$

$Y_{A[0,i]} \longleftarrow 1;$

else if $A[0,i] < A[1,i]$ then

$A[0, A[1,i]] \longleftarrow A[0,i];$

$A[1, A[1,i]] \longleftarrow A[0,i];$

$S[0, A[1,i]] \longleftarrow S[0,i] \oplus S[1,i];$

$S[1, A[1,i]] \longleftarrow S[0,i] \oplus S[1,i];$

$Y_i \longleftarrow 1;$

$Y_{A[1,i]} \longleftarrow 1;$

If Y_i=1

then if $A[0,i] > A[1,i]$ then $A[0,i] \longleftarrow A[1,i];$

$S[0,i] \longleftarrow S[1,i];$

else if $A[0,i] < A[1,i]$ then $A[1,i] \longleftarrow A[0,i];$

$S[1,i] \longleftarrow S[0,i].$

Figure 3.8 shows the final result of the example given in Figure 3.7 after an iteration of the algorithm.

3.2.4 Set up the states and determine π for next stage

When an iteration is completed, $A[p,i]$ will be the index of the representative of node i, and then each representative of the class will be assigned the value 0. As a result, the value of each variable b_i will be determined by the relative state bit, $S[p,i]$, in a straightforward manner.

For the example given above, we can now set the states to be:

Output switching elements:

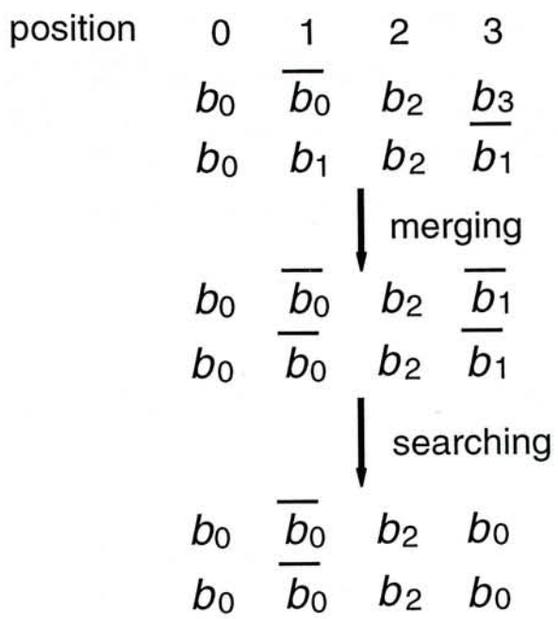


Figure 3.8: Merging and searching processes

$$b_0 = 0, \quad b_1 = \overline{b_0} = 1, \quad b_2 = 0, \quad b_3 = b_0 = 0$$

and to satisfy the symmetric routing constraint, we can set those

Input switching elements:

$$a_0 = b_2 = 0, \quad a_1 = \overline{b_0} = 1, \quad a_2 = b_0 = 0, \; a_3 = \overline{b_3} = 1$$

Note that one iteration of our algorithm will determine one routing bit (forward or reverse) for each input-output pair. To determine the next routing bit, we need to consider the input links and the output links of the middle stage of $B(n)$. As shown in Figure 3.9, the position of each link incident to the central subnetworks is the right circular shifting in address of the previous outcome. After shifting an address, the most significant bit, which is the previous routing bit, denotes which $B(n-1)$ (upper or lower) the link connects to, so it can be ignored in the route assignment of $B(n-1)$. Because $B_0(n-1)$ and $B_1(n-1)$ are independent.

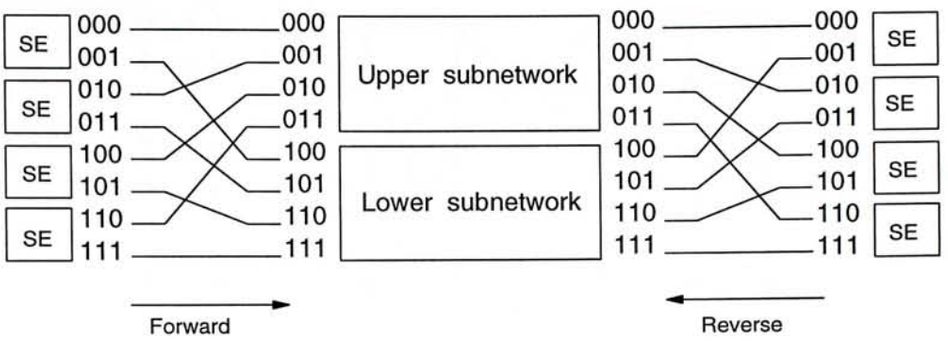


Figure 3.9: Link positions in the middle stage

Rewrite the π of the example given in **Subsection 3.1.3** into binary

$$\pi = \begin{pmatrix} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ 100 & 101 & 001 & XXX & 000 & 010 & 111 & 011 \end{pmatrix}$$

So the π_{next} will be

$$\pi_{next} = \begin{pmatrix} a_0 00 & \overline{a_0}00 & a_1 01 & \overline{a_1}01 & a_2 10 & \overline{a_2}10 & a_3 11 & \overline{a_3}11 \\ b_2 10 & \overline{b_2}10 & \overline{b_0}00 & XXX & b_0 00 & b_1 01 & \overline{b_3}11 & \overline{b_1}01 \end{pmatrix}$$

As the result of the algorithm

$$\pi_{next} = \begin{pmatrix} 000 & 100 & 101 & 001 & 010 & 110 & 111 & 011 \\ 010 & 110 & 100 & XXX & 000 & 101 & 111 & 001 \end{pmatrix}$$

or equivalently,

$$\pi_0 = \begin{pmatrix} 00 & 01 & 10 & 11 \\ 10 & XX & 00 & 01 \end{pmatrix}$$

$$\pi_1 = \begin{pmatrix} 00 & 01 & 10 & 11 \\ 10 & 00 & 01 & 11 \end{pmatrix}$$

where π_0 is the permutation of $B_0(2)$ and π_1 is the permutation of $B_1(2)$. The routing bits in the inner stages can be computed by our algorithm in a recursive manner. Refer to Figure 3.10 for the complete routing.

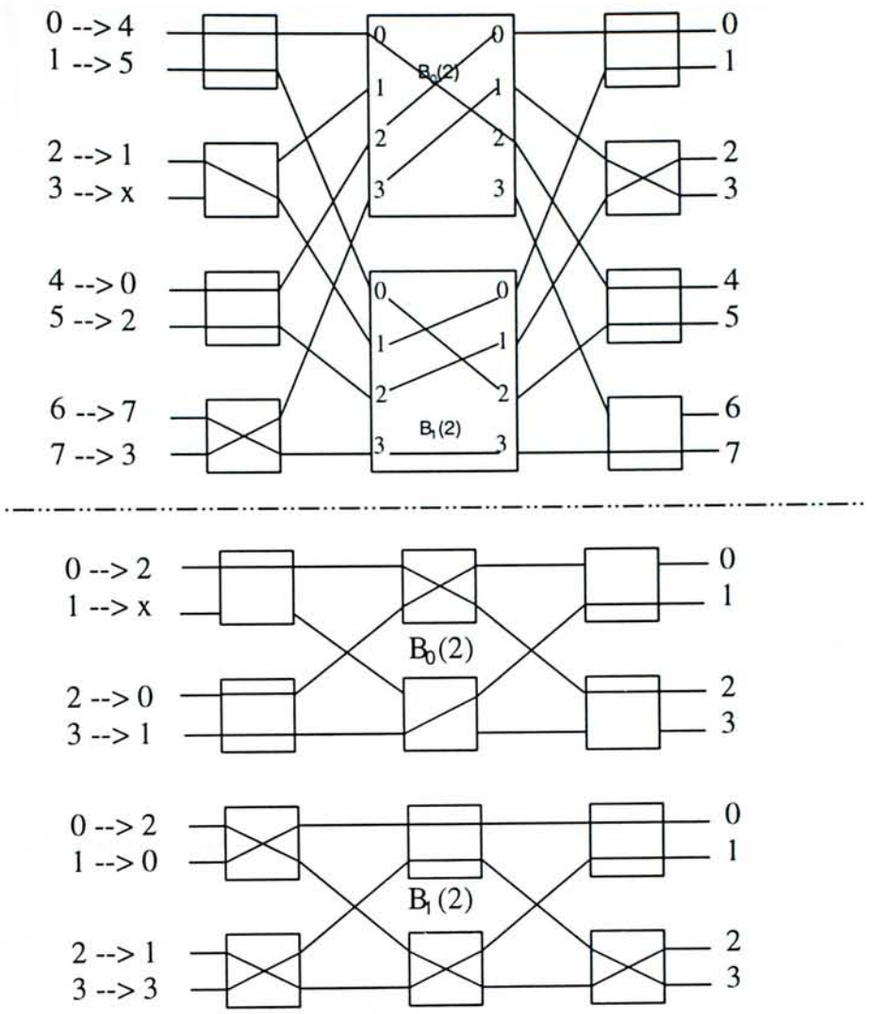


Figure 3.10: The route assignment calculated by the algorithm

3.2.5 Simulation results

Consider only one iteration of our algorithm, Figure 3.11 shows the simulation results. The y-axis represents the average number of executions needed to compute a routing bit for each input-output pair, where an execution consists of a step of searching and a step of merging. N is the network size. ρ is the probability that an input port is busy. When $\rho = 1$, representing the full permutation, the average number of executions is linear to $logN$. However, if there are some ports are idle, the curve seems to be presenting a function of $loglogN$.

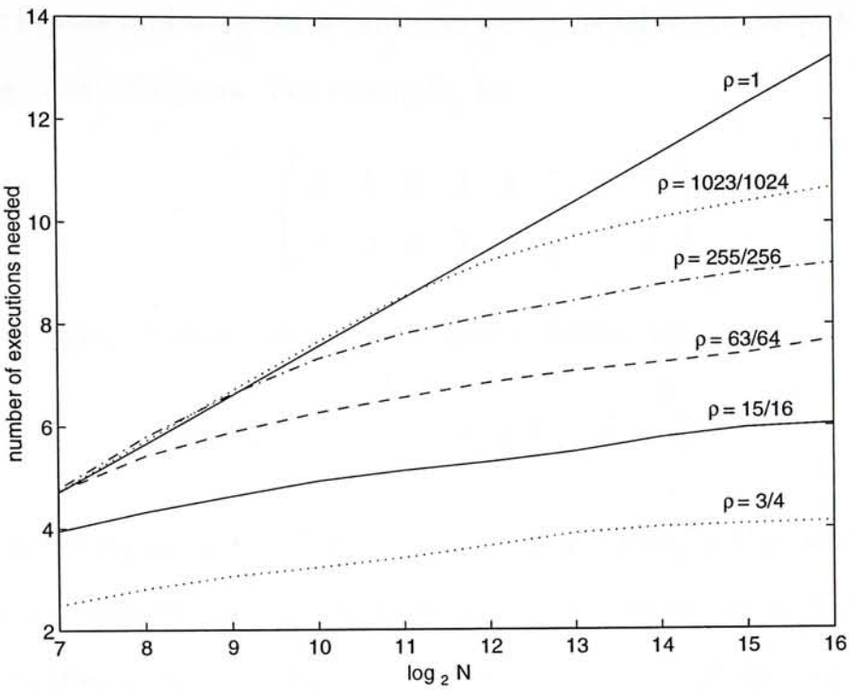


Figure 3.11: Simulation results

3.2.6 Time complexity

As the time complexity of the algorithm depends on the steps needed for propagation of searching, it follows from **Theorem 4**, the time needed for an iteration is $O(logN)$. Since there are $2n-1$ stages, we need to run the iteration $n = logN$ time. Consequently, the time complexity of the algorithm for total route assignment in Benes-Clos network is (log^2N).

3.3 Contention resolution

For any $k \in O$, if $\exists u, v \in I$ and $u \neq v$ such that $\pi(u) = \pi(v) = k$, then there is an **output contention** at output k. An switching element will encounter **contradiction** if neither state 0 nor state 1 can satisfy the two routing constraints. Output contentions will lead to such contradictions. Since our algorithm works

even if some inputs and outputs are idle, so we can resolve the output contentions by detecting contradictions. For example, let

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 3 & 0 & 3 & 4 & 1 & 3 & 3 \end{pmatrix}$$

The corresponding inequalities are obtained as following

$$\overline{b_0} \neq \overline{b_1}, \quad b_0 \neq \overline{b_1}, \quad b_2 \neq \overline{b_0}, \quad \overline{b_1} \neq \overline{b_1}$$

First, each inequality will be examined. For example the forth inequality ($\overline{b_1} \neq \overline{b_1}$) leads contradiction to routing constrains, so we must drop one of the connection requests, for example, we drop $\pi(7) = 3$ and let $\pi(7) = x$. The inequalities become

$$\overline{b_0} \neq \overline{b_1}, \quad b_0 \neq \overline{b_1}, \quad b_2 \neq \overline{b_0}, \quad \overline{b_1} \neq X$$

Notice that once a variable in an inequality is taken place by a don't care term X, this inequality will never cause contradiction and can be ignored in the following steps.

After that, we examine the inequalities two by two. The example above shows that $\overline{b_1}$ appears twice in the first ($\overline{b_0} \neq \overline{b_1}$) and the second ($b_0 \neq \overline{b_1}$) inequalities, again one of the connection requests must be dropped, for example let $\pi(3) = x$ yields the inequalities,

$$\overline{b_0} \neq \overline{b_1}, \quad b_0 \neq X, \quad b_2 \neq \overline{b_0}, \quad \overline{b_1} \neq X$$

At last, we check all of the four inequalities and let $\pi(5) = x$.

Notice that for any $N = 2^n$, the contradiction detection can be done within $n = log_2 N$ steps.

After the contradictions being detected and contentions being cleared in this stage, the π becomes

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 3 & 0 & x & 4 & x & 3 & x \end{pmatrix}$$

And the initializing equations are

$$\overline{b_0} = b_1, \quad b_0 = X, \quad b_2 = X, \quad b_1 = X$$

The result of the algorithm will be

$$b_0 = 0, \quad b_1 = \overline{b_0} = 1, \quad b_2 = 0, \quad b_3 = 0$$

The corresponding values of the variables a will be

$$a_0 = \overline{b_0} = 1, \quad a_1 = \overline{b_1} = 0, \quad a_2 = b_2 = 0, \quad a_3 = \overline{b_1} = 0$$

After an iteration, contentions may still exist and they can be cleared in the following stages in the same manner. This point is illustrated in Figure 3.12.

3.4 Algorithms applied to Clos network with 2^m central switches

An $N \times N$ rearrangeably non-blocking Clos network is shown in Figure 3.13. There are $2K$ switch modules distributed in the first and the last stages and

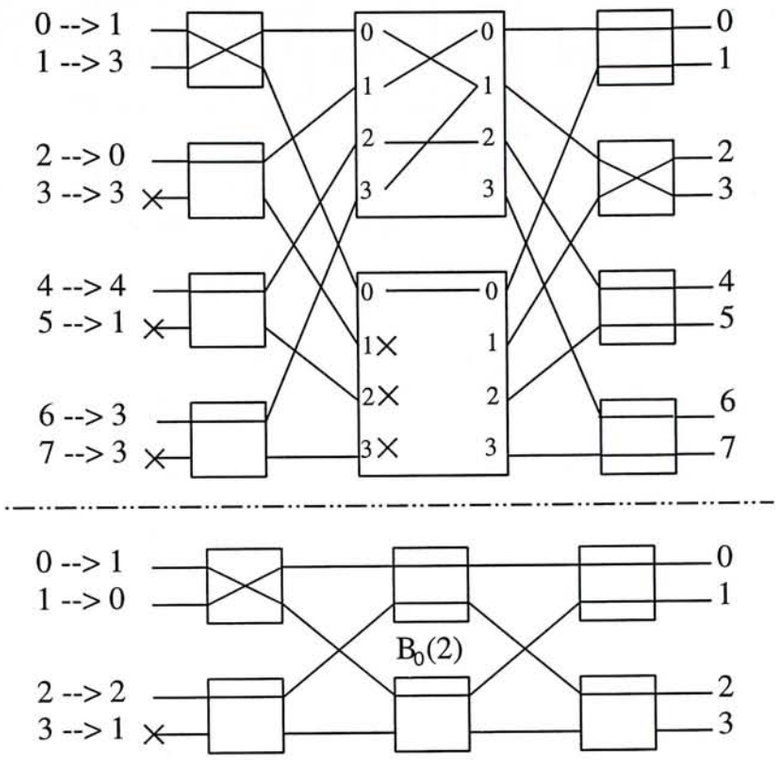


Figure 3.12: Contention resolution

each has $M \times M$ size, for $N = K \cdot M$. The other M switch modules of $K \times K$ size are in the middle stage.

If $M = 2^m$, we can label these M central modules by m binary bits such that in each $M \times M$ input (output) module, the *forward "destination address"* FDA (*reverse "destination address"* RDA) of a packet denotes a central module, or equivalently, a path to establish a connection. This is illustrated in Figure 3.14.

The two routing constraints in Benes network can be generalized to the three-stage Clos network as the following.

1. *Symmetric routing constraint* :

   If an input s and an output d are in a connection request, the FDA of s in the input module and the RDA of d in the output module must be the

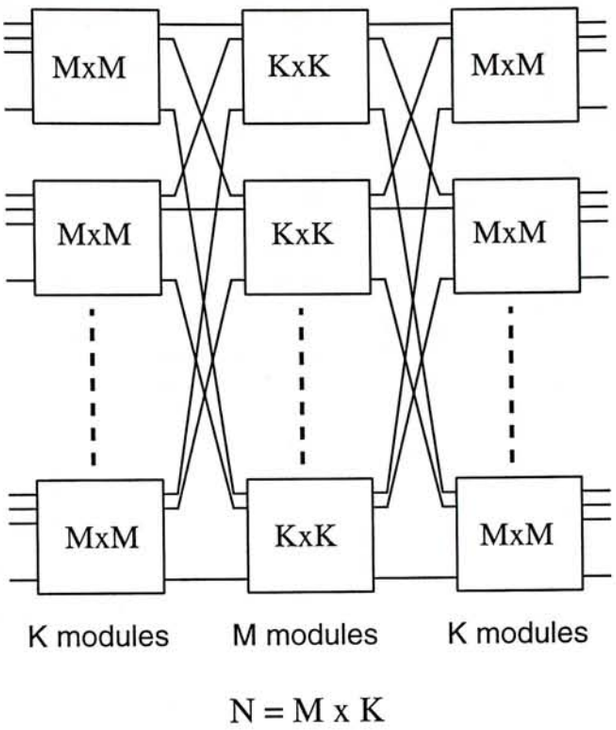


Figure 3.13: Three-stage Clos network

same.

2. *Internally conflict-free constraint* :

   Any two inputs (outputs) in the same input (output) module must be assigned to the different FDAs (RDAs).

In each input (output) module, the FDA (RDA) will be assigned to each packet bit by bit in order to satisfy the two routing constraints. This can be demonstrated by the following strategy. With reference to Figure 3.15, in each input (output) module, we partition all input ports (output ports) into $M/2$ groups, and each group consists of at most two ports. One of the input ports (output port) of each group will be assigned the value 0 as the most significant "destination address" bit and another will be assigned the value 1, in such a way that the input and the output in a connection request (s and d in the example)

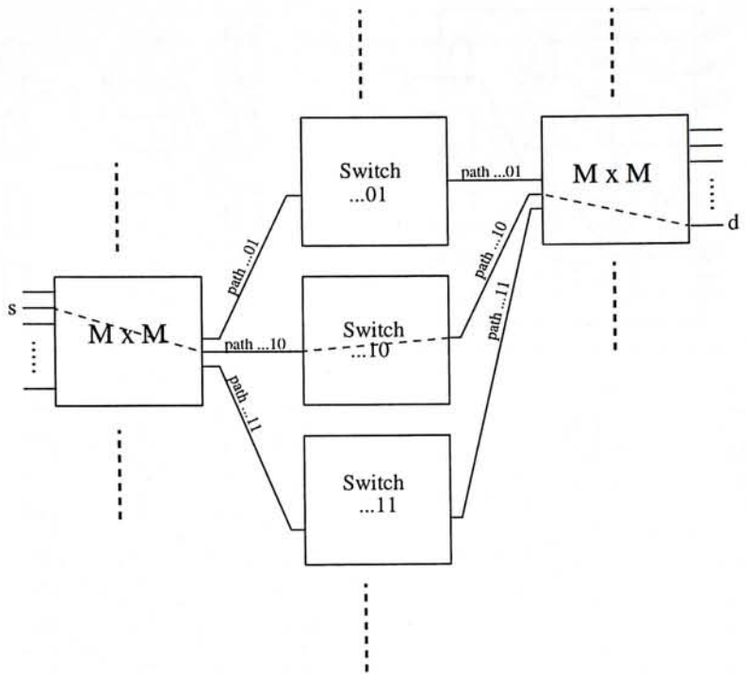


Figure 3.14: A "destination address" representing a path

must get the same binary value. The ports received 0 and the ports received 1 are independent, thus they can be separated before the second bits are assigned. After $k \leq m$ bits have been assigned, the packets will be partitioned into 2^k independent sets such that each set contains at most 2^{m-k} packets which have the same first k bits.

Compare to the operation of the first k and the last k stages in the Benes network, the basic principles are the same with the difference that, instead of setting the states of the 2×2 switching elements, we translate those binary bits into the "destination addresses".

Figure 3.16 demonstrates an explicit comparison between a 16×16 Benes network and a 16×16 Clos network with $2^2 = 4$ central modules. Each input (output) module in the Clos network can be regarded as a fictitious baseline

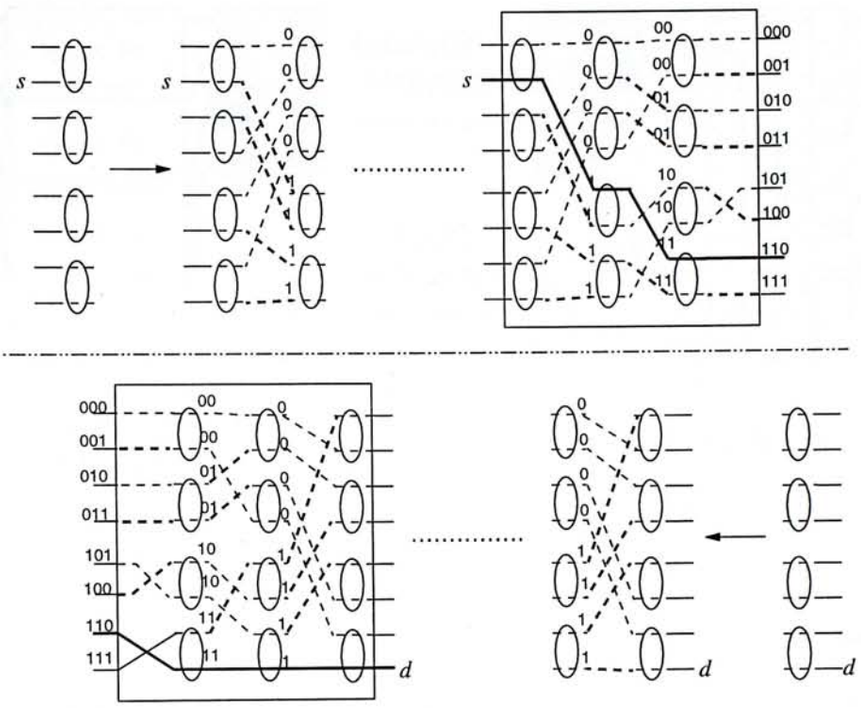


Figure 3.15: "Destination address" assignments in an input module

(reverse baseline) network. As far as the outer stages concerned, the Clos network with fictitious structure is isormophic with the Benes network. Thus, the connection requests in the Clos network with 2^m central modules can be mapped topologically to the Benes network, in such a way that the routing algorithm in Benes network can be applied. After m iterations, the route assignment will be returned to the Clos network. The time complexity is $O(logN \times logM)$.

3.5 Parallel algorithms in rearrangeability

In circuit switching configuration, new connection may request a path with the existence of old connections. If a path can always be established between any idle input and output, provided that the rearrangement of the old connections may be needed, then the network is said to be *rearrangeably non-blocking.* A sequential

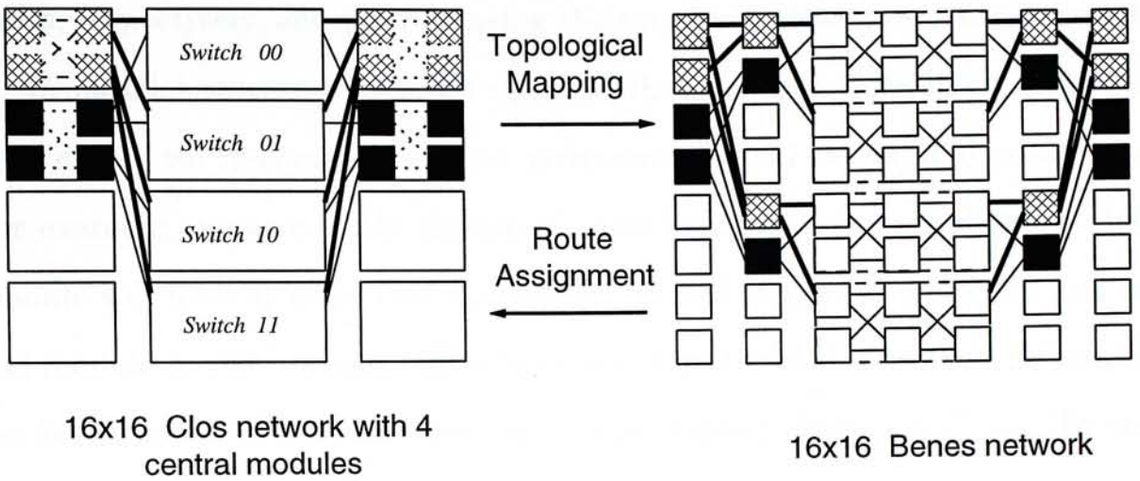


Figure 3.16: 16x16 Benes and fictitious structure of a 16x16 Clos networks

algorithm, called *looping algorithm*, was proposed to make the rearrangement in the three-stage non-blocking Clos network if it is necessary. In this section we are going to propose a parallel rearrangement set up algorithm based on the routing algorithm in Benes network.

With reference to the existing connections in a rearrangeably non-blocking Clos network shown in Figure 3.17 (a). Suppose now a new connection requests a path from input module i to output module j. Let C_i^I be the set of the central modules which has been used by input module i and C_j^O the set of central modules which has been used by output module j. In the example, $C_i^I = \{a, c\}$ and $C_j^O = \{b, c\}$. If $|C_i^I \bigcup C_j^O| < M$, where M is the total number of central modules, there currently exists at least one central module available to establish the new connection. If $|C_i^I \bigcup C_j^O| = M$, we can select two central modules, one is from $C_i^I - C_j^O$ and another from $C_j^O - C_i^I$, to make the rearrangement for the new connection. For example, if central module a and central module b are selected. The looping algorithm will make the rearrangement by searching an independent chain (i.e. a chain in Paull's matrix) composed by the connections related to a

and b, respectively, and interchanging their paths in these two central modules.

In parallel rearrangement set up algorithm, instead of finding an independent chain, we *re-compute* a route assignment for all the related connections. For example, because of the unique internal link property each input (output) module will have at most two connections related to central module a and central module b. All existing connections connected to these two central modules can be regarded as the Benes connections as shown in Figure 3.17 (b). To make the rearrangement, a set of connection requests in the fictitious Benes network can be found. In the example, we have

$$\pi' = \begin{pmatrix} 0 & 1 & \mathbf{2} & 3 & 4 & 5 \\ 4 & 3 & \mathbf{6} & 1 & 0 & x \end{pmatrix}$$

Thus, the parallel routing algorithm in Benes network can be applied. After the route assignment has been solved, it can be returned to the original Clos network. The result is shown in Figure 3.17 (c) and (d). Note that the connections related to other central modules (module c in the example) are not involved in the rearrangement.

Furthermore, our algorithm can also handle the rearrangement for more than one paths are requested at the same time, which is not allowed in looping algorithm. As shown in Figure 3.18, consider only those central modules with available paths for new connections and also all the existing connections related to these central modules, then the problem in rearrangeability can be reduced to the problem in route assignment among these central modules and the related connections. If we need to consider only M' of M central modules, then all related connections can be regarded as the connections in the Clos network with M' central modules . Note that if M' is not the power of two, we can choose

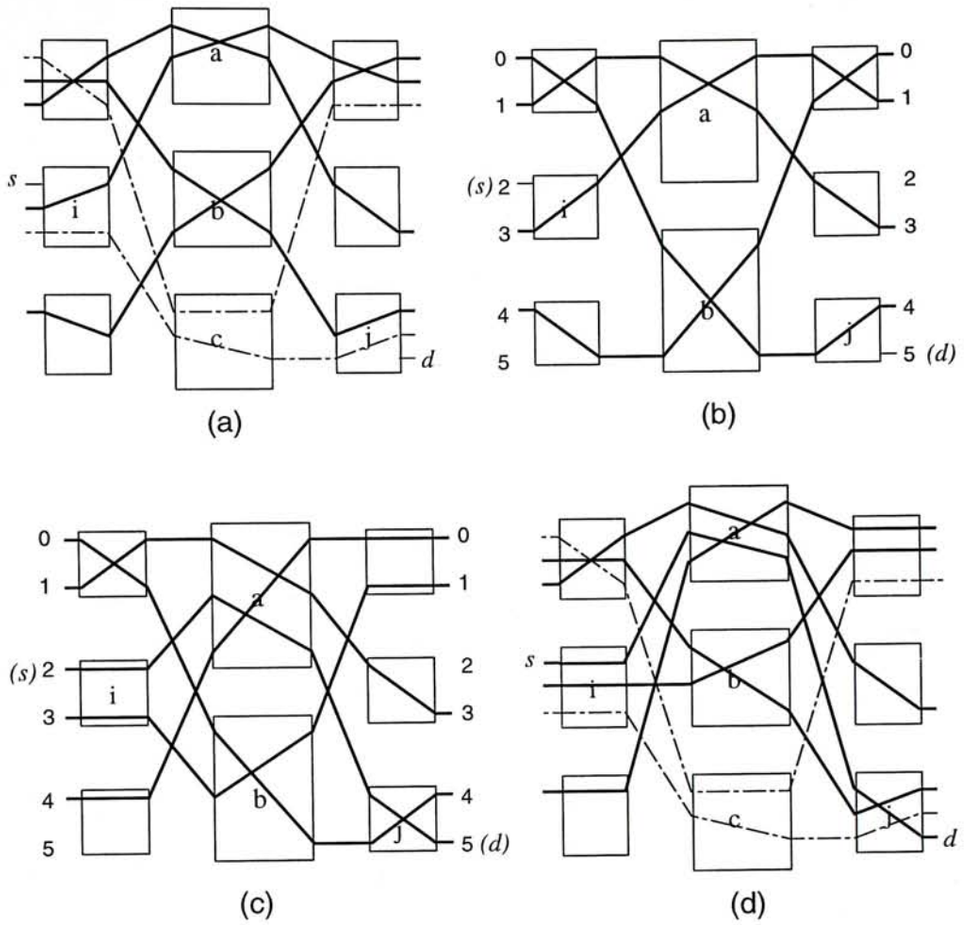
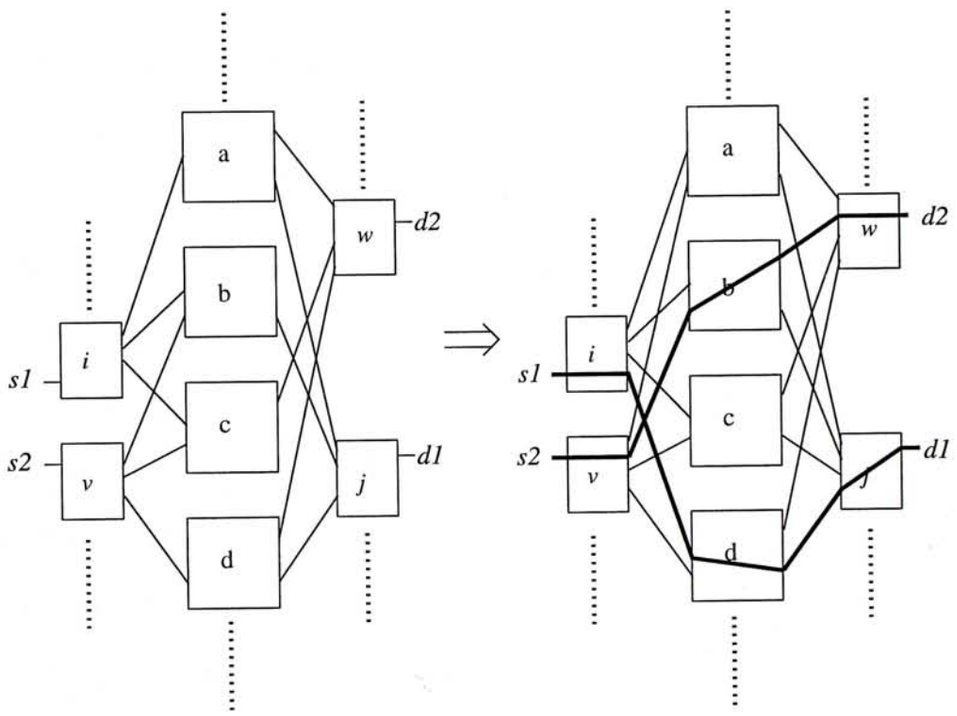


Figure 3.17: A rearrangement for the connection from input s to output d

some other central modules so that the total number of central modules involved in the calculation is the power of two.



A connection requests a path from i to j while another requests a path from v to w

Figure 3.18: A rearrangement for two new connection requests

Chapter 4

Conclusions

In this thesis we have investigated a new parallel routing algorithm based on the fundamental properties of Benes-Clos networks. In circuit switching system, a new connection may request a path in the network with the existence of old connections, however, the switching configuration can be rearranged at relatively low speed. In packet switching systems, switch fabrics must be able to provide internally conflict-free paths simultaneously and to accommodate packets requesting for connections in real-time as they arrive at the inputs. Thus, there is a necessity of parallel routing algorithm.

We formulate in Chapter 2 the routing problem in Benes-Clos networks to the problem of edge-coloring bipartite graphs, where the input modules and the output modules can be regarded as the vertices in the two disjoint sets respectively in a bipartite graph, and each central module can be regarded as an edge-color. In looping algorithm, if rearrangement is needed, an independent chain will be figured out and the two edge-colors on the chain will be interchanged so that the new colored edge can be set up. In parallel algorithm, the edge-coloring

will propagate from the representative of each class throughout that class at an exponential rate. To compare with the time complexity of sequential algorithm which is $O(N)$, parallel algorithm will finish an iteration within the time complexity of $O(logN)$.

In Chapter 3, two fundamental constrains in route assignment are derived from the structure of Benes-Clos networks: (1) Symmetric routing constrain; (2) Internally conflict-free constrain, where constrain (1) leads equalities and constrain (2) leads inequalities for a set of connection requests. In Benes network, these relationships can be represented by Boolean variables which denote the states of the 2×2 switching elements in the network. The searching procedure and the merging procedure, which perform a type of transitive closure, are proposed in this chapter to solve these equations. As far as routing is concerned, a Clos network with 2^m central modules is equivalent to a fictitious Benes network. Thus, our algorithm can also be applied to the Clos network in a straightforward manner. Furthermore, our algorithm can be applied to the rearrangeability of the Clos network. In this way, we not only can take the advantage of speeding up the computation, but also more than one connection requests can be processed at the same time.

Most existing parallel routing algorithms are not practical for packet switching because they either assume the set of connection requests is a full permutation or fail to deal with output contentions among the set of input packets. Also these algorithms always depend on tedious but unnecessary definitions or notations, this is because they were not derived directly from the basic observations on the structure of the three-stage Clos networks. Our parallel routing algorithm, which is presented in Chapter 3, can solve the routing problem naturally,

and also deal with partial permutation and output contention easily.

Finally, some related directions for further research are outlined in the following. An unsolved problem in this thesis is that if the number of the central modules is not the power of two, there is still no full parallel and optimized routing algorithm. Another important issue in packet switching is the capability of multicasting. It would be worthwhile to investigate the basic principle of the multicasting capability in Benes-Clos networks. Results and insights obtained in this thesis can possibly be extended to these topics.

Bibliography

[1] C. Clos, "A Study of Non-Blocking Switching Networks," *Bell Sys. Tech. Jour.*, Mar.1953,pp.406-424.

[2] V.E. Benes, "On Rearrangeable Three-Stage Connecting Networks," *Bell Sys.Tech. Jour.*, Sept. 1962, pp.1481-1492.

[3] G.F. Lev, N. Pippenger, L.G. Valiant, "A Fast Parallel Algorithm for Routing in Permutation Networks," *IEEE Tranc. Comput.*, Vol.C-30, No2, Feb. 1981, pp.93-100.

[4] D. Nassimi and S. Sahni, "Parallel Algorithms to Set Up the Benes Permutation Network," *IEEE Trans. Comput.*,Vol.C-31, Feb. 1982, pp.148-154.

[5] D.M. Koppelman and A.Y. Oruc, " A Self-Routing Permutation Network," *Jour. Parallel Distributed Comput.*, 1990, pp.140-151.

[6] D.C. Opferman and N.T. Tsao-Wu,"On a Class of Rearrangeable Switching Networks Part I: Control Algorithm," *Bell Sys. Tech. Jour.*, May-June 1971, pp.1579-1600.

[7] T.T. Lee and P.T. To, "Non-blocking and Self-routing Properties of Sort-Clos Networks," submitted to *International Journal of Computer Systems Science & Engineering* .

[8] T.Y. Feng and W. Young, "Parallel control algorithms for the reduced-omega-omega^{-1} network" *National Computer Conference*, 1985, pp.167-174.

[9] D. Nassimi and S. Sahni, " A Self-Routing Benes Network, and Parallel Permutation Algorithms," *IEEE Trans. Comput.*,Vol.C-30, May 1981, pp.157-161.

[10] B.G. Douglass and A.Y. Oruc, "On Self-routing in Clos Connection Networks," *IEEE Trans. Comm.*, Vol.41, No1,Jan. 1993, pp.121-124.

[11] C.S. Raghevandra and R.V. Boppana, "On Self-routing in Benes and Shuffle-exchange Networks," *IEEE Trans. Comput.*, Vol.40, No 9, Sept. 1991, pp.1057-1064.

[12] C.Y. Lee and A.Y. Oruc, " A Fast Parallel Algorithm for Routing Unicast Assignments in Benes Networks," *IEEE Trans. Parallel Distributed Sys.*, Vol.6, No 3, Mar. 1995, pp.329-333.

[13] D.S. Hochbaum, T. Nishizeki and D.B. Shmoys, " A Better than 'Best Possible' Algorithm to edge Color Multigraphs,"*Journal of Algorithms*, 7, 79-104, 1986, pp.79-103.

[14] J.Y. Hui, "Switching and Traffic Theory for Integrated Broadband Network," *Kluwer Academic Publishers.*

[15] R.J. Wilson, "Introduction to Graph Theory," *Longman Inc., New York*, 3rd edition.

[16] C. Berge, "The Theory of Graphs and Its Applications," *Methuen Wiley.*