A Nonbacktracking Matrix Decomposition Algorithm for Routing on Clos Networks

John D. Carpinelli, *Senior Member, IEEE,* and A. Yavuz Oruç, *Senior Member, IEEE*

***Abstract*— A number of matrix decomposition schemes were reported for routing on Clos switching networks. These schemes occasionally fail to find the right decomposition, unless backtracking is used. This paper shows that a partition may occur during the decomposition process, and that this is the underlying reason these algorithms fail for some decompositions. It then presents a parallel algorithm which can recognize when a partition exists and set up the Clos network without backtracking.**

I. Introduction

THREE-stage Clos networks have been studied extensively in telephone switching theory [2], [6]. A number of backtracking set-up or routing schemes have been reported in the literature for these networks [3], [10], [11], [16]. As shown in Fig. 1, a Clos network encompasses three stages where the first stage contains k switches, each of which has m inputs and n outputs. The second stage consists of n $k \times k$ switches, each of which receives exactly one input from each first-stage switch. By convention, the inputs to first-stage switch i $(1 \leq i \leq k)$ are numbered from $(i-1) \cdot m+1$ to $i \cdot m$. Each switch can realize any mapping of its inputs onto its outputs, provided that no input is mapped onto more than one output and that no output has more than one input mapped onto it. The final stage has k $n \times m$ switches, each of which derives one input from each second-stage switch. A Clos network with these parameters is referred to as an (n, m, k) Clos network. It is known that if $n \geq m$, the network is rearrangeable, and if $n \geq 2m-1$, the network is strictly nonblocking [6]. The number of inputs to the network is $N = m \cdot k$.

Routing is the process of setting the switches of a permutation network to realize a given permutation, or connection pattern from the inputs to the outputs. The looping algorithm is used frequently with Beneš networks [13], [18], but it does not generalize to all Clos networks. Andresen [1] developed an extension of the looping algorithm for Clos networks which have $m = n = 2^r$, where r is a positive integer. However, a different approach is needed to route on all Clos networks. One promising approach is the matrix decomposition class of routing algorithms.

Matrix decomposition algorithms make use of the design parameters of the Clos network. Most algorithms in the matrix decomposition class start by deriving a $k \times k$ matrix H_m, where $H_m[i, j]$ is the number of inputs to first-stage switch i which are to be routed to third-stage switch j under the permutation to be realized. Because each first-stage switch has m inputs, the sum of the entries in each row is m; since each last-stage switch has m outputs, the sum of the entries in each column is also m. All of these algorithms are based on a principle which extracts a permutation matrix, E_m, from H_m by forming the nonsingular matrix $H_{m-1} = H_m - E_m$, sets $m = m - 1$ and repeats the process until $m = 1$, at which time $H_1 = E_1$. The rationale behind this principle is the well-known Hall's theorem on systems of distinct representatives [8]. Consequently, by determining the setting for one second-stage switch, the problem of realizing a permutation on the original Clos network can be reduced to that of realizing a permutation on a Clos network with $m-1$ input switches in its outer stages. The same procedure can be applied until the remainder network reduces to a single second-stage switch.

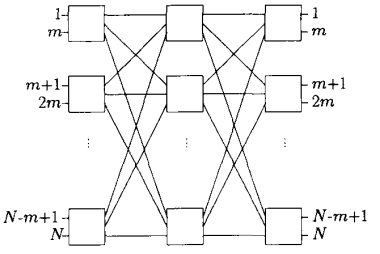


Fig. 1. Three-stage Clos network.

By extracting a permutation matrix from H_m, matrix decomposition algorithms do just that. The extracted matrix, E_m, defines the setting of one second-stage switch, and H_{m-1} is the setting of the $(n-1, m-1, k)$ Clos network. E_m also induces a partial setting for each first(last)-stage cell.

Most algorithms in the matrix decomposition class, with the exception of Neiman's algorithm [12], do not successfully realize all possible permutations. These algorithms do not recognize the inherent partitioning that exists within the matrices for some permutations, and create an extraction matrix with one or more rows of all zero elements. This is due to a condition which occurs in the H_m matrix during the decomposition procedure. As a result of this, matrix decomposition algorithms generally require *backtracking,* which may include and remove an element from E_m before the matrix is set. This is a bottleneck in matrix decomposition algorithms, and results in reduced routing speed. Ideally, a

Paper approved by J.M. Jaffe the Editor for Routing and Switching of the IEEE Communications Society. Manuscript received January 15, 1991; revised June 15, 1991.

J. D. Carpinelli is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102.

A. Y. Oruç is with the Department of Electrical Engineering, University of Maryland, College Park, MD 20740.

IEEE Log Number 9211211.

matrix decomposition algorithm should not use backtracking to route the Clos network. The remainder of the paper introduces the notion of partitioning which accounts for the failures of the previously reported matrix decomposition algorithms. It then gives a partitioning procedure which computes the E_m matrix without any backtracking.

II. Partitioning

Partitioning can best be defined using an example. Given a Clos network with $m = n = 4$ and $k = 5$, the permutation shown below in (1) has the matrix

$$H_4 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 1 & 1 \\ 1 & 2 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

A matrix decomposition algorithm would proceed by marking one nonzero entry, removing its row and column from the matrix, and repeating this on the remaining submatrix until it is null. The marked elements define the matrix to be extracted.

For example, assume that $H_4[1,1]$ is marked. Then the first row and first column are removed, and the submatrix left is

$$H_4 = \begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 2 & 1 & 1 \\ 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

An algorithm might attempt to mark $H_4[3,3]$, but this would result in the submatrix

$$H_4 = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Clearly, no permutation matrix can be derived from this matrix. The fault lies in the selection of $H_4[3,3]$. Once $H_4[1,1]$ is chosen, columns 4 and 5 have nonzero entries only in rows 2 and 3. Therefore, any elements marked in columns 4 and 5 must be from rows 2 and 3. More importantly, however, any elements marked in rows 2 and 3 must be from columns 4 and 5. If $H_4[3,3]$ is chosen, the only non-zero elements in columns 4 and 5 both appear in row 2. Since they cannot both be chosen, no permutation matrix can be derived which includes both $H_4[1,1]$ and $H_4[3,3]$. Hence, $H_4[3,3]$ is not a valid choice.

These remarks are generalized in the following theorem.

Theorem 1: Let $\mathcal{C}$ be any subset of nonzero columns of H_m, and let $\mathcal{R}$ be the set of all rows of H_m for which at least one column of $\mathcal{C}$ has a nonzero entry. If $|\mathcal{C}| = |\mathcal{R}|$, then an element in a row of $\mathcal{R}$ may be marked only if it is in a column of $\mathcal{C}$.

Proof: Assume that $|\mathcal{C}| = |\mathcal{R}| = x$, $i \in \mathcal{R}$, $j \notin \mathcal{C}$, and $H_m[i,j] \neq 0$. To prove the theorem by contradiction, assume that $H_m[i,j]$ is marked. Since no other element in row i may be marked, $\mathcal{R} \leftarrow \mathcal{R} - i$. Now $|\mathcal{R}| = x - 1$. Since the columns of $\mathcal{C}$ have nonzero elements only in the rows of $\mathcal{R}$, and only one element may be marked per row, at most $x - 1$ elements can be marked in the columns of $\mathcal{C}$. Therefore, at least one column of $\mathcal{C}$ will not have any marked elements, and no permutation matrix can be formed. Hence, $H_m[i,j]$ cannot be chosen, and the theorem is proved by contradiction. ■

The dual of this theorem is also true.

Theorem 2: Let $\mathcal{R}$ be any subset of nonzero rows of H_m, and let $\mathcal{C}$ be the set of all columns of H_m for which at least one row of $\mathcal{R}$ has a nonzero entry. If $|\mathcal{R}| = |\mathcal{C}|$, then an element in a column of $\mathcal{C}$ may be marked only if it is in a row of $\mathcal{R}$.

The proof is similar to that of the previous theorem. Note that any matrix which meets the conditions of Theorem 1 also meets the conditions of Theorem 2.

These two theorems can be combined into one theorem, which is based on a fact given by Shapiro [17].

Theorem 3: Given a matrix H_m, let P_1 and P_2 be $k \times k$ permutation matrices such that $P_1 H_m P_2 = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$, where A is $z \times z$, B is $z \times (k-z)$, C is $(k-z) \times z$, and D is $(k-z) \times (k-z)$. If there exist P_1 and P_2 such that B and/or C contain only zeros, then an element in H_m may be marked only if it is in either A or D in $P_1 H_m P_2$.

Proof: Since P_1 and P_2 are permutation matrices, $P_1 H_m P_2$ is just a series of row and column exchanges of H_m; entries are moved but not altered. If B contains all zeros, then $P_1 H_m P_2$ contains z rows which have nonzero elements only in exactly z columns. H_m must also contain z rows which have nonzero elements only in exactly z columns, and the conditions of Theorem 2 are met. If C contains all zeros, then $P_1 H_m P_2$ contains z columns which have nonzero elements only in exactly z rows, and the conditions of Theorem 1 are met. Note that if P_1 and P_2 can be chosen so that either B or C contains only zeros, then there exist P_1' and P_2' such that $P_1' H_m P_2'$ contain only zeros in the other quadrant of the matrix. ■

These three equivalent theorems divide H_m matrices into two classes: those which contain partitions and those which do not contain partitions.

In the above example, after $H_4[1,1]$ is marked, rows 4 and 5 have nonzero entries only in columns 2 and 3. By Theorem 2, an element in column 2 or 3 may be marked if and only if it is in row 4 or 5. Thus, $H_4[3,3]$ is not a valid choice. In effect, two submatrices are formed: one consisting of the rows and columns which meet the criteria of one of the theorems, and

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 1 & 5 & 13 & 17 & 14 & 15 & 18 & 19 & 9 & 10 & 16 & 20 & 2 & 6 & 7 & 11 & 3 & 4 & 8 & 12 \end{pmatrix} \quad (1)$$

one consisting of the remaining rows and columns. For the above example, after $H_4[1,1]$ is marked, the remaining matrix must be partitioned into

$$\begin{bmatrix} & 2 & 2 \\ & 1 & 1 \\ & & \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} & & \\ & & \\ 2 & 1 & \\ 2 & 1 & \end{bmatrix}.$$

Elements in these submatrices would be marked in the usual manner.

III. Why Jajszczyk's Algorithm Fails

Jajszczyk's algorithm [10] begins by setting up the matrix H_m in the usual manner. The number of zeros in each row and column is counted, and an arbitrarily chosen nonzero element in the row or column with the most zeros is marked. The row and column of the marked element are crossed out, the number of zeros is recounted, and the process is repeated until all rows and columns are crossed out. The matrix to be extracted is the permutation matrix with ones at the marked positions.

Consider the permutation below:

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 7 & 8 & 1 & 5 & 11 & 6 & 3 & 9 & 12 & 4 & 10 \end{pmatrix}.$$

The matrix for this permutation is

$$H_3 = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}.$$

Since two is the smallest number of nonzero elements in a row or column, any nonzero element in any row or column having this number of nonzero elements may be chosen. Arbitrarily, $H_3[1,3]$ is chosen and marked. Row 1 and column 3 are deleted from the matrix leaving

$$\begin{bmatrix} & & * & \\ 1 & 1 & & 1 \\ 1 & 1 & & 0 \\ 0 & 1 & & 2 \end{bmatrix}.$$

Column 4 has only two nonzero elements, so $H_3[4,4]$ is selected. The algorithm continues, marking $H_3[2,1]$ arbitrarily, and finally $H_3[3,2]$. The matrix, with the marked elements denoted by asterisks, is

$$H_3 = \begin{bmatrix} 1 & 0 & 2* & 0 \\ 1* & 1 & 0 & 1 \\ 1 & 1* & 1 & 0 \\ 0 & 1 & 0 & 2* \end{bmatrix}.$$

Cardot [3] provided the following counterexamples to Jajszczyk's algorithm. Given the matrix

$$H_4 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 \\ 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

the algorithm could mark the elements $H_4[3,6]$, $H_4[5,8]$, $H_4[6,1]$, and $H_4[8,2]$, in that order. After removing the rows and columns of the marked elements, the matrix becomes

$$H_4 = \begin{bmatrix} & 1 & 0 & 1 & & 0 & & 0 & 2 \\ & 0 & 0 & 1 & & 1 & & 0 & 0 \\ & & & & * & & & & \\ & 0 & 0 & 1 & & 0 & & 0 & 1 \\ & & & & & & * & & \\ * & & & & & & & & \\ & 1 & 1 & 0 & & 0 & & 2 & 0 \\ * & & & & & & & & \\ & 0 & 0 & 0 & & 2 & & 0 & 1 \\ & 0 & 0 & 1 & & 1 & & 0 & 0 \end{bmatrix}.$$

Obviously, choosing $H_4[7,4]$ or $H_4[7,9]$ is necessary, but either choice leaves a matrix with a column of zeros. This prevents a permutation matrix from being found.

Cardot's counterexample shows the flaw in Jajszczyk's algorithm after the second choice. After two passes the matrix becomes

$$H_4 = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & & 0 & & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & & 1 & & 0 & 0 \\ & & & & & & & & & \\ 1 & 0 & 0 & 0 & 1 & & 0 & & 0 & 1 \\ & & & & & & & & & \\ 2 & 0 & 2 & 0 & 0 & & 0 & & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & & 0 & & 2 & 0 \\ 0 & 2 & 0 & 1 & 0 & & 0 & & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & & 2 & & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 & & 1 & & 0 & 0 \end{bmatrix}.$$

At this point, rows 1, 2, 4, 6, and 9 have nonzero entries only in columns 1, 3, 5, 7, and 10 of the remainder of the matrix. Applying Theorem 2, let $\mathcal{R} = \{1,2,4,6,9\}$ and $\mathcal{C} = \{1,3,5,7,10\}$. Any element chosen from the columns of $\mathcal{C}$ must also be in the rows of $\mathcal{R}$. Specifically, the matrix must be partitioned into two submatrices: one consisting of the rows and columns of $\mathcal{R}$ and $\mathcal{C}$, and the other consisting of the other

rows and columns. This yields the submatrices

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ 0 & 1 & & 2 & \\ 2 & 1 & & 1 & \\ & & & & \\ 2 & 0 & & 0 & \end{bmatrix}.$$

The second submatrix must be furthered partitioned to

$$\begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & 1 & & 2 & \\ & 1 & & 1 & \\ & & & & \\ & & & & \end{bmatrix} \text{ and } \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ 2 & & & & \end{bmatrix}.$$

Obviously $H_m[10, 2]$ is a required choice, and $H_m[8, 2]$ is an illegal choice. Once $H_m[8, 2]$ is marked by Jajszczyk's algorithm, a permutation matrix can no longer be extracted. Note that Jajszczyk's algorithm could have chosen $H_m[10, 2]$ instead of $H_m[8, 2]$ and successfully found a permutation matrix without backtracking. The existence of a partition implies that an invalid choice *can* be made, but does not guarantee that it *will* be made.

This partitioning is not necessarily unique, but any valid partitioning will ultimately determine that $H_m[8, 2]$ is an invalid choice. This can easily be shown by contradiction. If $H_m[8, 2]$ is marked, the matrix becomes

$$H_4 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ & & & & * & & \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 2 & 0 \\ & * & & & & & \\ 1 & 0 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

In this matrix, rows 1, 2, 4, 6, 9, and 10 have nonzero entries only in columns 1, 3, 5, 7, and 10. Since only one entry may be chosen in each row and column, at most five elements may be marked in these six rows. Therefore, one row will not have any marked elements and a permutation matrix cannot be formed. Hence, $H_m[8, 2]$ is again an illegal choice.

Despite this shortcoming, Jajszczyk's algorithm provides an excellent strategy for making the arbitrary choices required by any matrix decomposition algorithm. It can make the matrix as sparse as possible, which leads to more partitioning later in the decomposition, and thus a faster routing. Jajszczyk's algorithm will work for all permutations if a mechanism to recognize and act upon partitions is included.

IV. Why Ramanujam's Algorithm Fails

Ramanujam [16] uses a different matrix than the other algorithms in this class, but one that is related to the standard H_m matrix. The *allocator matrix*, M, has dimension $k \times k$, and $M[i, j]$ is the set of all destinations of inputs to first-stage switch j which are output at third-stage switch i. It is actually the transpose of H_m, with the entries listed rather than counted.

The phase of the algorithm which extracts the desired matrix operates as follows. Set up an $k \times k$ matrix T, where $T[i, j]$ is the maximum element of $M[i, j]$, or 0 if $M[i, j]$ is empty. The largest element of T is marked, and its row and column are crossed off. This is repeated on the submatrix left in T until T is null or contains all zeros. If T is null, the marked elements define a matrix for extraction; these elements are deleted from M, and the process is repeated until M is null. If T is not null, reform T, replacing the largest value with a zero, and repeat this stage, choosing the next largest element of T. Finally, a renumbering is performed to accommodate the author's notation.

For example, consider the Clos network with $m = n = 3$ and $k = 4$. The permutation to be realized is

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 2 & 7 & 8 & 1 & 5 & 11 & 6 & 3 & 9 & 12 & 4 & 10 \end{pmatrix},$$

which is the same permutation used previously. The allocator matrix M and T matrix are

$$M = \begin{bmatrix} \{2\} & \{1\} & \{3\} & \Phi \\ \Phi & \{5\} & \{6\} & \{4\} \\ \{7,8\} & \Phi & \{9\} & \Phi \\ \Phi & \{11\} & \Phi & \{10,12\} \end{bmatrix} \quad \text{and}$$

$$T = \begin{bmatrix} 2 & 1 & 3 & 0 \\ 0 & 5 & 6 & 4 \\ 8 & 0 & 9 & 0 \\ 0 & 11 & 0 & 12 \end{bmatrix}.$$

Since 12 is the largest element of T, $T[4,4]$ is marked, and row 4 and column 4 of T are deleted. The largest remaining element is 9, so $T[3,3]$ is marked and its row and column are deleted. Continuing, $T[2,2] = 5$ and $T[1,1] = 2$ are chosen. Denoting each marked location with an asterisk, M becomes

$$M = \begin{bmatrix} \{2\}* & \{1\} & \{3\} & \Phi \\ \Phi & \{5\}* & \{6\} & \{4\} \\ \{7,8\} & \Phi & \{9\}* & \Phi \\ \Phi & \{11\} & \Phi & \{10,12\}* \end{bmatrix}.$$

Kubale [11] gives the following counterexample to Ramanujam's algorithm. Given a Clos network with $m = n = 2$ and $k = 4$, Ramanujam's matrix for the permutation

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 5 & 1 & 4 & 2 & 6 & 7 & 8 \end{pmatrix}$$

is

$$M = \begin{bmatrix} \Phi & \{1\} & \{2\} & \Phi \\ \{3\} & \{4\} & \Phi & \Phi \\ \{5\} & \Phi & \{6\} & \Phi \\ \Phi & \Phi & \Phi & \{7,8\} \end{bmatrix}.$$

The representation matrix is

$$T = \begin{bmatrix} 0 & 1 & 2 & 0 \\ 3 & 4 & 0 & 0 \\ 5 & 0 & 6 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix}.$$

Since 8 is the largest element, $T[4,4]$ is marked, and row 4 and column 4 are removed, leaving

$$T = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 0 \\ 5 & 0 & 6 \end{bmatrix}.$$

$T[3,3]$ would be chosen next, since 6 is the largest remaining element. T then becomes

$$T = \begin{bmatrix} 0 & 1 \\ 3 & 4 \end{bmatrix}.$$

Unfortunately, the algorithm then makes the only invalid choice, namely, $T[2,2]$. Since the process fails, the algorithm sets the largest element of the original T matrix to 0 and tries again. For this matrix, T then becomes

$$T = \begin{bmatrix} 0 & 1 & 2 & 0 \\ 3 & 4 & 0 & 0 \\ 5 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

No nonzero entry can be extracted from row 4 or column 4, and the algorithm cannot derive a matrix with exactly one nonzero entry in each row and column.

Ramanujam's matrix decomposition algorithm works for most, but not all, permutations. It fails when it zeroes out an entire row or column; this is due to its inability to deal with partitions within the matrix. Had partitioning been used in Kubale's counterexample, using any of the three theorems, the original representation matrix would have been partitioned into the submatrices

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 0 \\ 5 & 0 & 6 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} & & \\ & & \\ & & 8 \end{bmatrix}$$

Ramanujam's algorithm can handle this case once the matrix has been partitioned. Ramanujam's algorithm can handle any permutation if a mechanism to recognize and act upon inherent partitions is incorporated into it. The added mechanism is actually sufficient in itself; Ramanujam's algorithm would then serve as an heuristic to make arbitrary choices when no forced choice exists.

V. A Partitioning Algorithm

Any matrix which can cause backtracking in a matrix decomposition algorithm contains a partition. Any nonbacktracking algorithm to perform a matrix extraction must be able to either determine when a partition exists and act accordingly or ensure that no invalid choice is made if a partition exists. Neiman's algorithm acts upon partitions by convolving the marked elements until the partitions are accounted for although never recognized per se. An algorithm to recognize these partitions is given below.

PROCEDURE PARTITION (H_m, E_m);

var $H'_m, partition\_exists, M_1, M_2$;

BEGIN

1. $H'_m := H_m; E_m := 0$;
   WHILE $H'_m \neq \Phi$ DO
   BEGIN
2. $partition\_exists :=$ FALSE;
   GENERATE_PARTITION$(H'_m, partition_exits, M_1, M_2)$;
   IF $(partition\_exists =$ FALSE) THEN
3a. BEGIN
   Choose i, j such that $H'_m[i,j] \neq 0$;

$H'_m := H'_m \setminus i, j; E_m[i,j] := 1$
END
ELSE
3b. BEGIN
PARTITION(M_1, E_1);
PARTITION(M_2, E_2);
$E_m := E_m + E_1 + E_2; H'_m := \Phi$
END
END(while)
END(procedure);

The algorithm works as follows. Step 1 initializes the variables. The WHILE loop, consisting of steps 2, 3a, and 3b, adds elements to E_m until it is a permutation matrix. Step 2 calls subroutine GENERATE_PARTITION to check if a partition exists, i.e., if the conditions of Theorem 2 are met. If a partition exists, the subroutine forms the submatrices and returns them in M_1 and M_2. If no partition exists, step 3a marks an arbitrary nonzero element and removes its row and column from H'_m. If a partition does exist, step 3b recursively processes the two partition submatrices.

The heart of the algorithm is subroutine GENERATE_PARTITION. Here it is implemented as a highly parallel subroutine which checks all 2^k possible sets of rows of the matrix. This subroutine is shown below.

PROCEDURE GENERATE_PARTITION
$(H'_m, partition\_exists, M_1, M_2)$;

var $\mathcal{R}, \mathcal{C}$;

BEGIN

1. $M_1 := 0; M_2 := 0$;
PARFOR each possible set of rows of H'_m DO
BEGIN
2. $\mathcal{R} :=$ the set of rows of H'_m;
$\mathcal{C} :=$ the set of all columns of H'_m which have at least one nonzero element in a row of $\mathcal{R}$;
3. IF $|\mathcal{R}| = |\mathcal{C}|$ THEN
BEGIN
$M_1 :=$ the rows and columns of H'_m in $\mathcal{R}$ and $\mathcal{C}$;
$M_2 :=$ the rows and columns of H'_m not in $\mathcal{R}$ and $\mathcal{C}$;
$partition\_exists :=$ TRUE
END;
END(parfor)
END(procedure);

This subroutine generates all possible sets $\mathcal{R}$ in parallel, and checks all possible partitions. Step 1 sets the parameters and begins the parallel executions. Step 2 checks the conditions of Theorem 2. If a partition exists, step 3 forms the partition submatrices and sets *partition_exists*. Once a partition is found, **all** parallel executions are immediately terminated. The subroutine exits and returns the values derived from the parallel execution which finds the partition. If more than one partition is found, the algorithm arbitrarily selects one and returns its values in M_1 and M_2.

To illustrate the execution of this algorithm, consider the matrix

$$H_m = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}.$$

Step 1 sets $H'_m = H_m$ and $E_m = \mathbf{0}$. Step 2 calls subroutine GENERATE_PARTITION. Step 1 of the subroutine sets its parameters and begins the parallel executions. One of these executions has $\mathcal{R} = \{1,3\}$. Step 2 sets $\mathcal{C} = \{1,3\}$; since $|\mathcal{R}| = |\mathcal{C}| = 2$, step 3 sets *partition_exists* = TRUE, and

$$M_1 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \quad \text{and} \quad M_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

Returning to the main procedure, a partition does exist, so step 3b recursively processes the two submatrices. This may result in

$$E_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad E_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

The final result is

$$E_m = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

This algorithm will always generate the permutation matrix E_m without backtracking. This is because a partition is recognized *before* an element is chosen; thus, illegal choices are never made. The subroutine GENERATE_PARTITION is exhaustive, so a partition is always found if one exists.

The time complexity of PARTITION is dominated by the time complexity of subroutine GENERATE_PARTITION, which is called k times in the worst case. If no partition exists, the subroutine checks all 2^k possible sets of $\mathcal{R}$ the first time 2^{k-1} sets the second time, and so on. For all k iterations, this can result in at most $2^{k+1} - 1$ executions. Using one processor, the time complexity becomes $T(k) = O(2^k)$. This time can be improved using more than one processor; however, there are constraints. One element of E_m must be found before the next can be sought, so the k subroutine calls from PARTITION cannot overlap. The parallel branches in GENERATE_PARTITION are totally independent, so the time complexity of the subroutine can be reduced to $O(1)$ by using 2^k processors. Thus, using 2^k processors, the time complexity of PARTITION reduces to $O(k)$. The performance of PARTITION depends on the value of k. In particular, this algorithm is most efficient when $k = O(\lg N)$, where N is

the number of inputs to the Clos network. In this case, the time and hardware complexities of PARTITION are $O(\lg N)$, and $O(N)$, respectively. Hence, when $k = O(\lg N)$, one can decompose the H_m matrix without backtracking in logarithmic time and using a polynomial order of processors. As such, this fact shows that the conjecture by Bassalygo and Neiman [10] does not hold for all values of k. We also note that the worst time and hardware complexities of PARTITION are $O(N)$ and $O(2^N)$ which occur when $k = O(N)$.

VI. Conclusions

Partitioning is an important characteristic of the matrix decomposition class of routing algorithms for Clos networks. This paper has demonstrated its applicability and used it to improve current routing algorithms. A separate algorithm has been developed which makes use of partitioning to find the extraction matrix E_m.

In spite of the high hardware cost, the partitioning algorithm compares well to other algorithms in its class. Neiman's algorithm, the only other matrix decomposition algorithm which works for all permutations, has a runtime of $O(k^2m)$ for extracting one matrix [9]. The parallel version of the partitioning algorithm improves on this runtime, and does not require the backtracking used by Stage 2 of Neiman's algorithm. The algorithms of Jajszczyk and Ramanujam are both faster than the partitioning algorithm, but do not work on all permutations. The partitioning algorithm can be used to make these algorithms realize all permutations, or they can be used to generate the arbitrary choices of i and j in the partitioning algorithm.

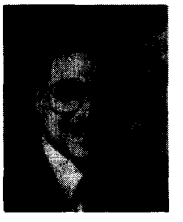
Finally, we should note that, after we submitted this paper, we found out from one of the referees that Gordon and Srikanthan [7] published an algorithm which they claimed to be nonbacktracking. However, Chiu and Siu [5] have reported an error in this algorithm. They also provided a modified version of the original algorithm which they claim to be correct based on simulation results, but without a proof. Unlike most matrix decomposition algorithms, the procedures in these papers use two matrices, rather than one, and are thus not directly amenable to the analysis presented in this paper.

Acknowledgment

The authors thank the anonymous referees for their constructive remarks and suggestions.

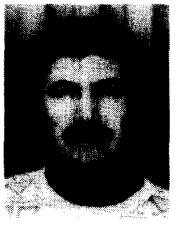
References

[1] S. Andresen, "The looping algorithm extended to base 2^t rearrangeable switching networks," *IEEE Trans. Commun.*, vol. COM-25, pp. 1057–1063, Oct. 1977.
[2] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic.* New York: Academic, 1965.
[3] C. Cardot, "Comments on 'A simple control algorithm for the control of rearrangeable switching networks,'" *IEEE Trans. Commun.*, vol. COM-34, p. 395, Apr. 1986.
[4] J. Carpinelli, "Interconnection networks: Improve routing methods for Clos and Beneš networks," Ph.D. dissertation, Rensselaer Polytech. Inst., Troy, NY, Aug. 1987.
[5] Y. K. Chiu and W. C. Siu, "Comment: Novel algorithm for Clos-type networks," *Electron. Lett.*, vol. 27, no. 6, pp. 524–526, Mar. 1991.
[6] C. Clos, "A study of non-blocking switching networks," *Bell Syst. Tech. J.*, vol. 32, no. 2, pp. 406–424, Mar. 1953.
[7] J. Gordon and S. Srikanthan, "Novel algorithm for Clos-type networks," *Electron. Lett.*, vol. 26, no. 21, pp. 1772–1774, Oct. 1990.
[8] P. Hall, "On representatives of subsets," *J. London Math. Soc.*, vol. 10, pp. 26–30, 1935.
[9] F. Hwang, "Control algorithms for rearrangeable Clos networks," *IEEE Trans. Commun.*, vol. COM-31, pp. 952–954, Aug. 1983.
[10] A. Jajszczyk, "A simple algorithm for the control of rearrangeable switching networks," *IEEE Trans. Commun.*, vol. COM-33, pp. 169–171, Feb. 1985.
[11] M. Kubale, "Comments on 'Decomposition of permutation networks,'" *IEEE Trans. Comput.*, vol. C-31, p. 265, Mar. 1982.
[12] V. I. Neiman, "Structure et command optimales de reseaux de connexion sans blocage," *Ann. Telecommun.*, vol. 24, pp. 232–238, July–Aug. 1969.
[13] D. C. Opferman and N. T. Tsao-Wu, "On a class of rearrangeable switching networks: Part I: Control algorithm," *Bell Syst. Tech. J.*, vol. 50, no. 5, pp. 1579–1600, May–June 1971.
[14] O. Ore, *The Four-Color Problem.* New York: Academic, 1967.
[15] A. Y. Oruç, "Designing cellular permutation networks through coset decompositions of symmetric groups," *J. Parallel Distributed Comput.*, vol. 4, no. 2, pp. 404–422, Aug. 1987.
[16] H. R. Ramanujam, "Decomposition of permutation networks," *IEEE Trans. Computers*, vol. C-22, pp. 639–643, July 1973.
[17] L. W. Shapiro, *Introduction to Abstract Algebra.* New York: McGraw-Hill, 1975.
[18] A. Waksman, "A permutation network," *J. ACM*, vol. 15, no. 1, pp. 159–163, Jan. 1968.



John D. Carpinelli (S'81–M'83–SM'92) received the B.E. degree in electrical engineering from Stevens Institute of Technology, Hoboken, NJ, in 1983, and the M.E. degree in electrical engineering and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1984 and 1987, respectively.

In 1986 he joined the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology, Newark, where he is currently an Associate Professor. His primary research interests include interconnection networks, multiprocessor system design, and biomedical computing.



A. Yavuz Oruç (S'81–M'83–SM'92) received the B.Sc. degree in electrical engineering from Middle East Technical University in 1976, the M.Sc. degree in electronics from University of Wales in 1978, and the Ph.D. degree in electrical engineering from Syracuse University in 1983.

He was a faculty member in the Electrical, Computer and Systems Engineering Department at Rensselaer Polytechnic Institute, Troy, NY, from 1983 to 1987. He is currently a faculty member in the Electrical Engineering Department at the University of Maryland, College Park. His research interests include advanced computer systems, parallel processing, and interconnection networks.

Dr. Oruç is a member of the IEEE Computer and Information Theory Societies.