Brief Contributions

Optimally Routing LC Permutations on *k*-Extra-Stage Cube-Type Networks

Qing Hu, Xiaojun Shen, and Weifa Liang

Abstract—It is difficult to partition an arbitrary permutation into a minimum number of groups such that conflict-free paths for all source-destination pairs in each group can be established on an omega network. Based on linear algebra theory, this paper presents an optimal algorithm which solves this problem for the LC class of permutations on a large class of multi-stage networks. This algorithm extends the previous result which deals with the BPC class of permutations on the omega network.

Index Terms—*k*-extra-stage cube-type networks, LC permutations, multistage interconnection networks, optimization problems, permutation realization, .

1 Introduction

An $N \times N$ ($N = 2^n$) multistage interconnection network (MIN) provides connections among N processors. The class of multistage cube-type networks (MCTN) refers to those MINs which are topologically equivalent to the Generalized Cube, including Omega, Baseline, Indirect Cube, Regular SW Banyan with S = F = 2, etc. [3]. An MCTN is implemented by $n = logN$ stages of 2×2 switching elements (SE) with each stage having $N/2$ SEs. By setting each SE to either "cross" or "through" state, an MCTN can provide a unique path between any source to any desired destination. Two paths will conflict if they meet at a common output port of a commen SE. The communications among processors are often represented by an N-permutation π which defines a unique destination processor d for each source processor s, $d = \pi(s)$. To support efficient parallel computations, one would like an MIN to be able to provide the N connections (N paths) simultaneously. An N-permutation is said to be admissible to (or realizable on) an MCTN if N conflict-free paths exist. We refer to this problem as the **PA** (*permutation admissibility*) problem. A closely related problem is: If a given permutation is not admissible, how should the N pairs be divided into a minimum number of groups (passes) such that the conflict-free paths for all pairs in each group can be established simultaneously. We refer to this problem as the **MP** (*minimum number of passes*) problem. It is easy to see that the **PA** problem is a subproblem of the **MP** problem.

A k-Extra-stage MCTN (k-EMCTN) ($k > 1$) is obtained by adding k more stages in front of an MCTN, which provides 2^k disjoint paths between any pair of source and destination [7], [8]. When k increases, its fault-tolerance ability increases, and so does the permutation capability. An MCTN can be viewed as a 0-EMCTN. Fig. 1 illustrates a 1-Extra-stage omega network and the two paths for the source-destination pair (1000, 1011).

Discussions of the **PA** problem on MCTNs can be found in [9], [10], [11]. The **MP** problem was first raised by Wu and Feng in [3]. Raghavendra and Varmar [12] presented an $O(N \log N)$ algorithm to solve the **MP** problem for the class of BPC (*bit permutation complement*) permutations on an omega network (denoted by Ω). Shen [13] extended this result to k-omega network (k-Ω). Whether the **MP** problem for an arbitrary permutation on a k-EMCTN (even $k = 0$) can be solved in polynomial time remains an open problem. In this paper, we solve the **MP** problem for a more general class of permutations, the *LC* (*linear combination and complement*) class, on k-EMCTNs. The LC class includes much more permutations than the BPC class does (BPC and other frequently used classes are all subclasses of LC [15]) and is useful in the parallel access of data arrays [14]. In this paper, we present an $O(N \log N)$ algorithm which realizes an arbitrary LC permutation on any k-Ω ($0 \le k \le n-1$) with a minimum number of passes. Our discussion will be concentrated on the omega network. Using the concept of equivalence [3], we can readily deal with the other k-EMCTNs by finding the corresponding permutation on the k-Ω.

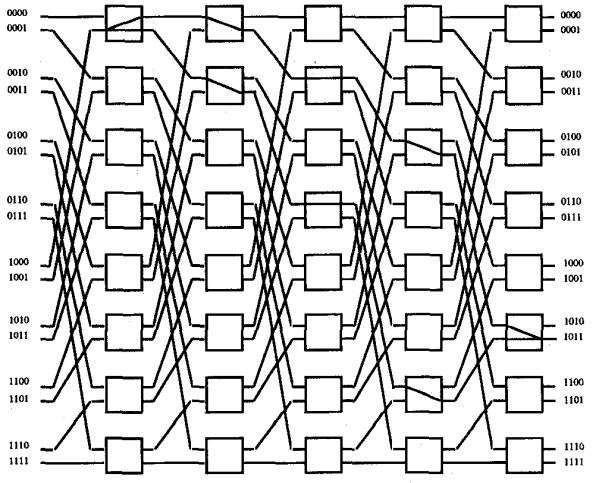

Fig. 1. A 1-omega network and the two disjoint paths between 1000 and 1011.

2 Preliminaries

2.1 Linear Algebra Concepts

Our algorithm will use some linear algebra notations. Let us introduce the linear space S_m first. Let $\mathcal{GF}(2)$ be the field ($\{0, 1\}, \oplus, \bullet$), where $\oplus$ and $\bullet$ are module two binary addition and multiplication respectively. S_m is defined as the set of all m-dimensional binary vectors, $S_m = \{ (a_0, a_1, ..., a_{m-1})^T \mid a_0, a_1, ..., a_{m-1} \in \mathcal{GF}(2) \}$. We define operation "+" on S_m as $(a_0, a_1, ..., a_{m-1})^T + (b_0, b_1, ..., b_{m-1})^T = (a_0 \oplus b_0, a_1 \oplus b_1, ..., a_{m-1} \oplus b_{m-1})^T$. Thus, $(S_m, +)$ is a Linear Space on $\mathcal{GF}(2)$, where $\mathbf{0} = (0, 0, ..., 0)^T$ is the 0-vector of $(S_m, +)$ and $-\mathbf{x} = \mathbf{x}$ for any $\mathbf{x} \in S_m$.

A subset U of S_m, $U = \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{k-1}\} \subseteq S_m$, is said to be *linearly independent* if the equation

$$\sum_{i=0}^{k-1} \alpha_i \mathbf{u}_i = \alpha_0 \mathbf{u}_0 + \alpha_1 \mathbf{u}_1 + \ ... \ + \alpha_{k-1} \mathbf{u}_{k-1} = \mathbf{0}$$

implies $\alpha_0 = \alpha_1 = ... = \alpha_{k-1} = 0$, otherwise, U is *linearly dependent*. A vector $\mathbf{v}$ is linearly independent of U, if $\beta v + \sum_{i=0}^{k-1} \alpha_i \mathbf{u}_i = 0$ implies $\beta = 0$ (denoted by $\mathbf{v} \perp U$), otherwise, $\mathbf{v}$ is linearly dependent on U or linearly representable by U (denoted by $\mathbf{v} \propto U$). $\mathbf{v} \propto U$ if and only if $\mathbf{v}$ can be represented by a linear combination of vectors in

• *The authors are with the Computer Science Telecommuni ations Program, University of Missouri-Kansas City, 5100 Rockhill Rd., Kansas City, MO 64110. E-mail: xshen@ctsp.umkc.edu.*

Manuscript received Nov. 10, 1993; revised May 31, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95119.

U, $v = \sum_{i=0}^{k-1} \alpha_i \mathbf{u}_i$. A vector set $V = \{\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{h-1}\}$ is linearly independent of U, if $\sum_{i=0}^{h-1} \beta_i \mathbf{v}_i + \sum_{i=0}^{k-1} \alpha_i \mathbf{u}_i = 0$ implies $\beta_0 = \beta_1 = ... = \beta_{h-1} = 0$ (denoted by $V \perp U$). Two vector sets U and V are *linearly equivalent*, if all vectors in U can be linearly represented by vectors in V and vice versa.

A *base* of U is a maximal linearly independent subset of U, denoted by base(U), and its cardinarity is called the *rank* of U, denoted by rank(U). The vectors of a base are called *base vectors*. Let rank(U) = r and base(U) = $\{\mathbf{b}_0, \mathbf{b}_1, ..., \mathbf{b}_{r-1}\}$ be a base of U. For each $\mathbf{u}_i \in U$, $\mathbf{u}_i = \sum_{j=0}^{r-1} \alpha_{ij} \mathbf{b}_j$, $i = 0, 1, ..., k-1$. Obviously, each vector $\mathbf{u}_i$ of U corresponds to a unique r-dimensional vector $(\alpha_{i0}, \alpha_{i1}, ..., \alpha_{i,r-1})$ in S_r which is called the coordinate of $\mathbf{u}_i$ under base(U). In linear algebra theory, it is well–known that a subset of U is linearly independent if and only if the set of coordinates under some base is linearly independent.

Given a vector set U, $U = \{u_0, u_1, ..., u_{k-1}\} \subseteq Sm$, let m(U) denote the matrix whose *ith* column is $\mathbf{u}_i$. For example, if

$$U = \left\{\begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}\right\}, \text{ then } m(U) = \begin{pmatrix}1&1&0\\0&1&0\\1&0&1\end{pmatrix}.$$

Now, we define another relation, a relation among the row vectors of m(U). For $\mathbf{v} \in S_m$, let v(i) denote the *ith* component of v. Given a $U \subseteq S_m$, the *ith* row sequence of U, denoted by $\mathbf{r}_i(U)$, is defined as the new vector $(u_0(i), u_1(i), ..., u_{k-1}(i))$ which is the *ith* row of m(U) and r(U) denotes the set of row vectors of m(U), i.e., $r(U) = \{\mathbf{r}_i(U) \mid i = 0, 1, ..., m-1\}$. Note that $|r(U)| \le m$, since $\mathbf{r}_i(U) = \mathbf{r}_j(U)$ $(i \neq j)$ is possible. Two vector sets U and W are said to be *horizontally equivalent*, if, for any $0 \le i, j \le m-1$, $\mathbf{r}_i(U) = \mathbf{r}_j(U) \Leftrightarrow \mathbf{r}_i(W) = \mathbf{r}_j(W)$.

LEMMA 2.1. *Given any two vector sets U, $W \subseteq S_m$, if U and W are linearly equivalent then U and W are horizontally equivalent.*

PROOF. Suppose U and W are linearly equivalent. Let base(U) be a base of U and rank(U) = |base(U)| = r. Let $W = \{\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_{k-1}\}$, $r \le k$. Then each $\mathbf{w}_i$ $(0 \le i \le k-1)$ is a linear combination of vectors in base(U) = U. Let $\alpha_i = (\alpha_{i0}, \alpha_{i1}, ..., \alpha_{i,r-1})^T$ $(0 \le i \le k-1)$ be the coordinate of $\mathbf{w}_i$. We have m(W) = m(U)M, where M is an $r \times k$ matrix $[\alpha_0, \alpha_1, ..., \alpha_{k-1}]$. Since $\mathbf{r}_h(U) = \mathbf{r}_i(U)$ implies $\mathbf{r}_h(\text{base}(U)) = \mathbf{r}_i(\text{base}(U))$, $\mathbf{r}_h(W) = \mathbf{r}_h(\text{base}(U))M = \mathbf{r}_i(\text{base}(U))M = \mathbf{r}_i(W)$.

Thus, $\mathbf{r}_h(W) = \mathbf{r}_i(W)$. Similarly, we can find a matrix M' such that m(U) = m(W)M'. Therefore, if $\mathbf{r}_h(W) = \mathbf{r}_i(W)$, then we have $\mathbf{r}_h(U) = \mathbf{r}_i(U)$. □

Given two vector sets U and V, if $V \perp U$, many interesting properties can be derived. In the following we list some propositions that are very useful to our routing algorithm. For simplicity, we assume $V \perp U$; base(U) is a base of U; and we use small letters to represent vectors. These propositions can be readily verified by anyone possessing a basic knowledge of linear algebra [18].

PROPOSITION 2.1.

i) V is linearly independent, and rank(V) = |V|;
ii) rank(U ∪ V) = rank(U) + rank(V) = rank(U) + |V|.

PROPOSITION 2.2.

i) $V \perp U$ if and only if $V \perp U \cup \{y\}$, where $\mathbf{y} \propto U$;
ii) $V \perp U$ if and only if $V \perp$ base(U).
iii) if U' is linearly equivalent to U, $V \perp U'$.

PROPOSITION 2.3.

i) If $\mathbf{w} = \mathbf{v} + \mathbf{y}$ where $\mathbf{v} \in V$ and $\mathbf{y} \propto (V - \{\mathbf{v}\}) \cup U$, then $\mathbf{w} \perp (V - \{\mathbf{v}\}) \cup U$;
ii) If $\mathbf{z} \perp U \cup V$ and $\mathbf{w} = \mathbf{v} + \mathbf{y}$ where $\mathbf{v} \in V$ and $\mathbf{y} \propto (V - \{\mathbf{v}\}) \cup U$, then $\mathbf{w} \perp (V - \{\mathbf{v}\}) \cup \{\mathbf{v} + \mathbf{z}\} \cup U$.

PROPOSITION 2.4. For any $\mathbf{w} \perp U \cup V$, $V \cup \{\mathbf{w}\} \perp U$ and $V \perp U \cup \{\mathbf{w}\}$ (A method to extend vector set V or U).

PROPOSITION 2.5.

i) If $\mathbf{u} \in U$, $\mathbf{v} \in V$, then $(V - \{\mathbf{v}\}) \cup \{\mathbf{v} + \mathbf{u}\} \perp U$;
ii) If $\mathbf{w} \perp U \cup V$, $\mathbf{v} \in V$, then $(V - \{\mathbf{v}\}) \cup \{\mathbf{v} + \mathbf{w}\} \perp U$;
iii) If $\mathbf{u} \in U$, $\mathbf{v} \in V$, and $\mathbf{u} \perp U - \{\mathbf{u}\}$, $V \perp (U - \{\mathbf{u}\}) \cup \{\mathbf{u} + \mathbf{v}\}$.

2.2 A Representation of Permutations

An N-permutation π can be represented by the following table (called transition matrix):

Source	Destination
$s_{0,0}\, s_{0,1} \cdots s_{0,n-1}$	$d_{0,0}\, d_{0,1} \cdots d_{0,n-1}$
$s_{1,0}\, s_{1,1} \cdots s_{1,n-1}$	$d_{1,0}\, d_{1,1} \cdots d_{1,n-1}$
... ...	
$s_{i,0}\, s_{i,1} \cdots s_{i,n-1}$	$d_{i,0}\, d_{i,1} \cdots d_{i,n-1}$
... ...	
$s_{N-1,0}\, s_{N-1,1} \cdots s_{N-1,n-1}$	$d_{N-1,0}\, d_{N-1,1} \cdots d_{N-1,n-1}$

where $s_{0,0}\, s_{0,1} \cdots s_{0,n-1}$ is the binary representation of i and $d_{i,0}\, d_{i,1} \cdots d_{i,n-1}$ is the binary representation of $\pi(i)$, $i = 0, 1, ..., N-1$. Let $S_j = (s_{0,j}, s_{1,j}, ..., s_{N-1,j})^T$, $j = 0, 1, ..., n-1$ and $D_j = (d_{0,j}, d_{1,j}, ..., d_{N-1,j})^T$, $j = 0, 1, ..., n-1$, any permutation can be represented by a transition sequence as follows [13]: $S_0, S_1, ..., S_{n-1}, D_0, D_1, ..., D_{n-1}$, where $S_0, S_1, ..., S_{n-1}$ are constant vectors without changing with different permutations, while $D_0, D_1, ..., D_{n-1}$ depend on the given permutation. For simplicity, we will use notation $\pi = (S_0, S_1, ..., S_{n-1}, D_0, D_1, ..., D_{n-1})$ in the later discussions. The *linear closure* of $S_0, S_1, ..., S_{n-1}$, denoted by $\mathcal{G}_n$, is defined as

$$\mathcal{G}_n = \left\{\sum_{i=0}^{n-1} \alpha_i S_i \mid \alpha_i \in \mathcal{GF}(2),\ i = 0, 1, ..., n-1\right\}.$$

Obviously, $\text{rank}(\mathcal{G}_n) = n$, and $S = \{S_0, S_1, ..., S_{n-1}\}$ is a base of $\mathcal{G}_n$. For any $U \subseteq \mathcal{G}_n$, we say U satisfies the *condition of distinctness*, if $i \neq j \Leftrightarrow \mathbf{r}_i(U) \neq \mathbf{r}_j(U)$. By the definition, it is easy to see that S satisfies the condition of distinctness.

LEMMA 2.2. *For any $U \subseteq \mathcal{G}_n$, rank(U) = n if and only if U satisfies the condition of distinctness.*

PROOF. Since $\text{rank}(\mathcal{G}_n) = n$, rank(U) = n implies that U is another base of $\mathcal{G}_n$. Any two bases are linearly equivalent, i.e., U and S are linearly equivalent. By Lemma 2.1, they are also horizontally equivalent. Therefore, U satisfies the condition of distinctness. Now we assume U satisfies the condition of distinctness, which implies that |r(U)| is equal to the number of components in a vector of U, i.e., |r(U)| = N. On the other hand, there is a bijection between |r(U)| and the set of |U|-bit binary numbers which has a total of $2^{|U|}$ different numbers, therefore, $|r(U)| \le 2^{|U|}$. By Lemma 2.1, |r(U)| = |r(base(U))|. If |base(U)| = r < n, then |r(U)| = |r(base(U))| = $2^r < 2^n = N$, a contradiction. Thus, the lemma holds. □

Let $\mathbf{1} = (1, 1, ..., 1)^T \in S_N$. A complement of a vector $\mathbf{u}$ is defined as $\mathbf{u} + \mathbf{1}$, i.e., the binary complement on each component of the vector. $\mathcal{G}'_n$ is defined as the extension of $\mathcal{G}_n$ such that $\mathcal{G}'_n$ consists of the linear combination of S and/or their complement, i.e.,

$$\mathcal{G}'_n = \left\{ \sum_{i=0}^{n-1} \alpha_i S_i + \alpha_n \mathbf{1} | \alpha_i \in \mathcal{GF}(2), i = 0, 1, \ldots, n \right\}.$$

Similarly, we define the extension of S as $S' = \{S_i + \alpha\mathbf{1} \mid \alpha \in \mathcal{GF}(2), i = 0, 1, \ldots, n-1\}$. Let $U' = \{\mathbf{u}'_0, \mathbf{u}'_1, \ldots, \mathbf{u}'_{r-1}\}$ be a subset of $\mathcal{G}'_n$. There exists a unique subset of $\mathcal{G}_n$, $U = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{r-1}\}$ such that

$$u_i = \begin{cases} u'_i & u'_i \in \mathcal{G}_n \\ u'_i + 1 & u'_i \notin \mathcal{G}_n \end{cases}$$

LEMMA 2.3. *U and U' are horizontally equivalent.*

PROOF. Assume $\mathbf{r}_h(U) = \mathbf{r}_l(U)$, i.e., $u_i(h) = u_i(l)$, $i = 0, 1, \ldots, r-1$. If $u'_i = u_i$, then $u'_i(h) = u'_i(l)$, otherwise,

$$u'_i(h) = u_i(h) \oplus 1 = u_i(l) \oplus 1 = u'_i(l),$$

too. Thus, $u'_i(h) = u'_i(l)$ is true for all $i = 0, 1, \ldots, r-1$, i.e., $\mathbf{r}_h(U') = \mathbf{r}_l(U')$. On the other hand, if $\mathbf{r}_h(U') = \mathbf{r}_l(U')$, then $\mathbf{r}_h(U) = \mathbf{r}_l(U)$. □

By the definition of permutation, $\{D_0, D_1, \ldots, D_{n-1}\}$ satisfies the condition of distinctness. Moreover, adding different constraints on $D_0, D_1, \ldots, D_{n-1}$ will produce different subclasses of the N-permutations, for example,

BP: $D_i \in S$, $i = 0, 1, \ldots, n-1$; **BPC**: $D_i \in S'$, $i = 0, 1, \ldots, n-1$.
L: $D_i \in \mathcal{G}_n$, $i = 0, 1, \ldots, n-1$; **LC**: $D_i \in \mathcal{G}'_n$, $i = 0, 1, \ldots, n-1$.

3 An Outline of the Optimal Routing Algorithm

The window method is a well-known method to solve many routing problems on the omega network (Ω) [11], [13]. Given an N-permutation π with the transition sequence $S_0, S_1, \ldots, S_{n-1}, D_0, D_1, \ldots, D_{n-1}$, an Ω-window is defined as a set of n consecutive vectors in this transition sequence. For example, the *i*th Ω-window is $\Omega\_W_i = \{S_i, S_{i+1}, \ldots, S_{n-1}, D_0, \ldots, D_{i-1}\}$, $i = 0, 1, \ldots, n-1$. Two paths of a permutation conflict if they use two input ports of a common SE at a certain stage and request the same exit port. In that case, these two paths have the same transition address in the corresponding window, i.e., that window does not satisfy the condition of distinctness. By Lemma 2.2, the rank of this window must be less than n. To apply the window method to a *k*-EMCTN, we need to insert *k* variable vectors into the transition sequence between S_{n-1} and D_0. As a result, we obtain a new transition sequence:

$$S_0, S_1, \ldots, S_{n-1}, \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, D_0, D_1, \ldots, D_{n-1},$$

where $\mathbf{x}_i$ corresponds to the setting of SEs at stage $(-k + i)$. Unlike S_i and D_i, which are determined by the permutation, the values of $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$ will be determined by the algorithm, and these values will affect the "number of passes." In a k-Ω, we have $n + k + 1$, k-Ω_windows,

$$k\text{-}\Omega\_W_i = \{S_{i+k}, S_{i+k+1}, \ldots, S_{n-1}, \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, D_0, \ldots, D_{i-1}\}, \quad i = -k, \ldots, -1, 0, 1, \ldots, n.$$

The **PA** problems on an Ω and a k-Ω correspond to the following two propositions respectively.

PROPOSITION 3.1. A permutation is admissible on an Ω if and only if $\text{rank}(W_i) = n$ for any i, $0 \le i \le n$.

PROPOSITION 3.2. A permutation is admissible on a k-Ω if and only if there are k vectors $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}$ such that $\text{rank}(W_i) = n$ for all i, $-k \le i \le n$.

If a given N-permutation is not admissible on a k-Ω, then the **MP** problem arises. From proposition 3.2, we are going to partition these N paths into a minimum number of groups, such that, if path h and l are in the same group, $\mathbf{r}_h(W_i) \neq \mathbf{r}_l(W_i)$ for any i, $-k \le i \le n$. We present two lemmas first.

LEMMA 3.1. *Given an N-permutation π and a k-Ω, let $W_i = W'_i \cup X_i$ be defined as above, and*

$$d_{\min} = MIN\{rank(W'_i) \mid i = -k, \ldots, -1, 0, 1, n\}.$$

The minimum number of passes of π on k-Ω is lower bounded by $2^{n-d_{min}-k}$.

PROOF. Without loss of generality, assume $\text{rank}(W'_i) = d_{\min}$. *It* implies $\text{rank}(W_i) \le d_{\min} + k$. Thus, W_i has at most $2^{d_{\min}+k}$ distinct row sequences, i.e., at least $2^{d_{\min}+k}$ paths have the same transition address at window W_i, and they must be assigned in different groups. □

Given an N-permutation $\pi = (S_0, S_1, \ldots, S_{n-1}, D_0, D_1, \ldots, D_{n-1}) \in LC$, there exists a unique $\pi^{\sim} = (S_0, S_1, \ldots, S_{n-1}, \tilde{D_1}, \ldots, \tilde{D_{n-1}}) \in L$ such that

$$\tilde{D_i} = \begin{cases} D_i & D_i \in \mathcal{G}_n \\ D_i + 1 & D_i \notin \mathcal{G}_n \end{cases}$$

LEMMA 3.2. *A solution to the **MP** problem for π is also a solution to the **MP** problem for $\pi^{\sim}$, and vice versa.*

PROOF. Let W_i and $W_i^{\sim}$ be the *i*th windows of π and $\pi^{\sim}$, respectively. By Lemma 2.3, W_i and $W_i^{\sim}$ are horizontally equivalent for $i = -k, \ldots, 0, 1, \ldots, n-1$. Path$(s_1, \pi(s_1))$ and path$(s_2, \pi(s_2))$ conflict if and only if path$(s_1, \pi^{\sim}(s_1))$ and path$(s_2, \pi^{\sim}(s_2))$ conflict. Thus, a solution of **MP** problem for π is also a solution for $\pi^{\sim}$, and vice versa. □

We will show that $2^{n-d_{\min}-k}$ groups is enough for any LC permutation. Based on Lemma 3.2, we will discuss the **MP** problem for class L only. An outline of our partition algorithm is as follows:

Let $\pi = (S_0, S_1, \ldots, S_{n-1}, D_0, D_1, \ldots, D_{n-1}) \in L$, $X = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}\}$ be the k variable vectors, W_i, W'_i, and X_i be defined as above. Let $t = n - k - d_{\min}$, where $d_{\min} = \text{MIN}\{\text{rank}(W'_i) \mid i = -k, \ldots, -1, 0, 1, \ldots, n\}$. We assign each $\mathbf{x}_i$ a vector of $\mathcal{G}_n$, and find a set of t vectors from $\mathcal{G}_n$, $OR = \{\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{t-1}\}$, such that the following condition i) is satisfied for $i = k, \ldots, -1, 0, 1, \ldots, n$:

$$\begin{cases} \text{i)} & X_i \perp W'_i; \\ \text{ii)} & \text{rank}(W_i \cup OR) = n. \end{cases}$$

Since $\mathbf{r}_p \in \mathcal{G}_n$, $0 \le p \le t-1$, $\mathbf{r}_p$ can be represented by $\sum_{j=0}^{n-1} \alpha_{p,j} S_j$. Thus, we define a function $f(r_p, i) = \bigoplus_{j=0}^{n-1} \alpha_{p,j} S_{i,j}$, where $s_{i,j}$ is the *j*th bit of the binary representation of i. Then, we construct a weight function ϕ for each path $(i, \pi(i)) \in \pi$ such that

$$\phi(i, \pi(i)) = \sum_{p=0}^{t-1} f(r_p, i) 2^p = \sum_{p=0}^{t-1} \left(\bigoplus_{j=0}^{n-1} \alpha_{p,j} s_{i,j} \right) 2^p.$$

Now, the SEs on $n + k$ stages are set by the transition sequence $S_0, S_1, \ldots, S_{n-1}, \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, D_0, D_1, \ldots, D_{n-1}$, and the paths with the same weight are classified into the same group. Since $0 \le \phi((i, \pi(i))) \le 2^t - 1$, all paths are partitioned into 2^t groups.

THEOREM 3.1. *The above partition algorithm correctly solves the **MP** problem for π.*

PROOF. Since $t = n - d_{\min} - k$, by Lemma 3.1, the number of groups has already reached its lower bound. We only need to show any two paths within the same group do not conflict with each

other. Note that $(f(r_0, i), f(r_1, i), ..., f(r_{t-1}, i))$ is the *ith* row sequence of the vector set OR. By the definitions, we have the following assertion: for any $-k \le i \le n-1$,

i) if two paths p and q conflict in window i, then $r_p(W_i) = r_q(W_i)$;
ii) if two paths p and q have the same weight, then $r_p(OR) = r_q(OR)$.

Since $\text{rank}(W_i \cup OR) = n$, by Lemma 2.2, $W_i \cup OR$ satisfies the condition of distinctness, i.e., $r_p(W_i \cup OR) \neq r_q(W_i \cup OR)$ for $p \neq q$, which implies that no two paths with the same weight conflict in window i. □

4 The Detailed Procedure

The key step of our algorithm is to construct the vector set X and OR such that condition (I) is satisfied. The construction is done by the procedure *Assignment* which is given below. The basic idea of procedure Assignment is as follows: First, let OR be a set of t 0-vectors and X be the first k vectors which are removed from the first k windows. It is easy to see that such construction makes the first k windows satisfy the condition i). Then, check the rest of $n+1$ windows one by one and update at most one vector of either X or OR each time when the condition i) is no longer satisfied. For example, if $X \perp W'_{i-1}$ but $X \perp W'_i$ is not true, then we will modify one (only one) vector in X to obtain X^* such that $X^* \perp W'_i$, too. Note: It is very important that such modification do not affect the previous windows, i.e., $X^* \perp W'_j (j \le i \le 1)$.

The procedure Assignment will use the following two functions:

1) Function cv(V, U : vector set; **w** : vector) : vector. This function outputs a vector **v** such that ($\mathbf{w} = \mathbf{v} + \mathbf{y}$) and ($\mathbf{v} \in V$) and ($\mathbf{y} \propto (V - \{\mathbf{v}\}) \cup U$), where V, U, **w** are input variables. If such a **v** doesn't exist, then return **0**.
2) Function ids(U, V : vector set) : vector set. This function returns the maximum subset of U which is linearly independent of V, i.e., ids(U,V) = U' such that ($U' \subseteq U$) and ($U' \perp V$) and ($\mathbf{u} \propto U' \cup V$ for any $\mathbf{u} \in U$).

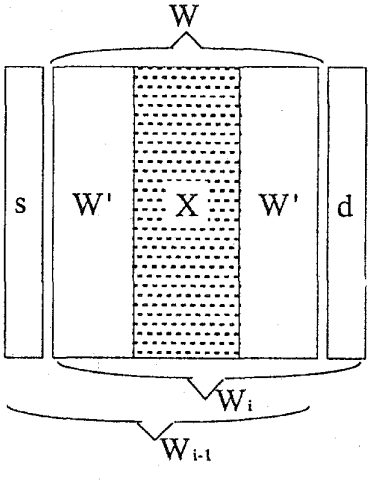

Fig. 2. An illustration for the algorithm.

```
Procedure Assignment.
Let  S={S0, S1, ..., Sn-1},
     D={D0, D1, ..., Dn-1},
     X={x0, x1, ..., xk-1}, and
     OR={r0, r1, ..., rt-1}.

for j = 0 to t-1
do  rj:=0;

for i=-k to -1
do  {xi+k := removed vector of window i};

for i:=0 to n-1 do
     begin Let s be the removed vector and
       d be the adding vector for
       obtaining Wi  from Wi-1, as
       shown in Fig. 2.
       W := Wi-1 - {s},
       W':= W'i-1-{s} ;

     do case
     case 1.  s ⊥ W, d ⊥ W:
     if rank(W ∪ OR ∪ {d})<n
     then
       {TempS := ids(OR, W);
            r := cv(TempS, W, d);
           OR := (OR-{r}) ∪ {r + s}; }

     case 2.  s ⊥ W, d ∝ W:
     if d ⊥ W'
     then
       {x := cv(X,W',d);
          X := (X-{x}) ∪ {x + s}; }
     else
     if rank(W ∪ OR ∪ {d})< n
     then
       {TempS := ids(OR, W);
          let r ∈ OR-TempS;
             OR := (OR - {r}) ∪ {r + s}; }

     case 3.  s ∝ W, d ⊥ W: do nothing;

     case 4.  s ∝ W, d ∝ W:
     if d ⊥ W'
     then
       {TempS :=ids(OR, W);
          let u ∈ TempS;
              x := cv(X, W', d);
              X := (X - {x})∪{x + u}; }
end;
```

Let $x_p^{(i)} (p = 0, 1, ..., k-1)$ and $r_p^{(i)} (p = 0, 1, ..., h-1)$ be the results after W_i has been considered in the procedure Assignment. $X^{(i)}$ denotes

$$\left\{x_p^{(i)} | p = 0, 1, ..., k-1\right\}$$

and $OR^{(i)}$ denotes

$$\left\{r_p^{(i)} | p = 0, 1, ..., t-1\right\}.$$

Following the above notations,

$$W_j^{(i)} = W'_j \cup X^{(i)} (j = -k, ..., 0, 1, ..., i)$$

Let Cond(I)[i, j] denote the following conditions:

i) $X_j^{(i)} \perp W'_j$ where $W'_j = W_j - X_j^{(i)}$;
ii) $\text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) = n$,

Condition i) in the above partition algorithm corresponds to the following group of conditions: Cond(I)[i, j] ($-k \le j \le n$). In the procedure Assignment, Cond(I)[n, j] ($-k \le j \le n$) always remains true as i increases from $-k$ to n, which is shown in the following lemma.

LEMMA 4.1. *In the procedure Assignment, after W_i has been considered, conditions Cond(I)[i ,j] ($-k \le j \le i$) remain true for i= –k, ..., 0, 1, ..., n.*

PROOF. We prove it by induction on i. The cases for $i \le 0$ are obviously true, which serve as the inductive basis. Now let us consider the case of i for $i > 0$. The inductive assumption is that all Cond(I)[i –1, j] (j = 0, 1, ..., i – 1) are true. Now we are going to prove that any Cond(I)[i, j] (j = 0, 1, ..., i) is also true.

Let us introduce some notations first. The *ith* iteration of the algorithm may update a vector in X or OR to make $X^{(i-1)}$ or $OR^{(i-1)}$ differ from the result $X^{(i)}$ or $OR^{(i)}$. Correspondingly, we have

$$W' = W'_{i-1} - \{\mathbf{s}\} = W'_i - \{\mathbf{d}\},$$

$$W^{(i-1)} = X^{(i-1)} \cup W' = W^{(i-1)}_{i-1} - \{\mathbf{s}\},$$

and

$$W^{(i)} = X^{(i)} \cup W' = W^{(i)}_i - \{\mathbf{d}\}$$

Besides, we use $OR_j^{(i-1)}$ to denote $\text{ids}\left(OR^{(i-1)}, W_j^{(i-1)}\right)$ and $OR_j^{(i)}$ to denote $\text{ids}\left(OR^{(i)}, W_j^{(i)}\right)$ for $j \le i$. Note that, if

$$X^{(i-1)} = X^{(i)} \text{ and } OR^{(i-1)} = OR^{(i)},$$

Cond(I)[i, j] is assured by Cond(I)[i – 1, j] for $j \le i - 1$; if $X^{(i-1)} = X^{(i)}$, $X_j^{(i)} \perp W'_j$ is also assured by $X_j^{(i-1)} \perp W'_j$ for $j \le i - 1$. Therefore, we need to pay attention to the cases of $X^{(i-1)} \ne X^{(i)}$ and $OR^{(i-1)} \ne OR^{(i)}$. We distinguish among those cases according to the algorithm.

CASE 1. According to the algorithm,

$$X^{(i)} = X^{(i-1)} \text{ and } W^{(i)} = W^{(i-1)},$$

which implies that $X_j^{(i)} \perp W'_j$ for $j \le i - 1$ by the inductive assumption. $X_i^{(i)} \perp W'_i$ is proved by Proposition 2.4. We prove that $\text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) = n$ for $j \le i$. If

$$\text{rank}(W^{(i-1)} \cup OR^{(i-1)} \cup \{\mathbf{d}\}) = n,$$

we just set

$$OR^{(i-1)} = OR^{(i)}.$$

Moreover, $W_j^{(i)} = W^{(i)} \cup \{\mathbf{d}\} = W^{(i-1)} \cup \{\mathbf{d}\}$, the inductive assumption implies $\text{rank}\left(W_i^{(i)} \cup OR^{(i)}\right) = n$. Thus, Cond(I)[i, j] is true for any $j \le i$.

Now, we assume $\text{rank}(W^{(i-1)} \cup OR^{(i-1)} \cup \{\mathbf{d}\}) < n$. Note that $\text{rank}(W \cup OR \cup \{\mathbf{d}\}) < n$ in the algorithm is equivalent to

$$\text{rank}(W^{(i-1)} \cup OR^{(i-1)} \cup \{\mathbf{d}\}) < n.$$

By the inductive assumption, $\text{rank}\left(W_{i-1}^{(i-1)} \cup OR^{(i-1)}\right) = n$, i.e., $\text{rank}(W^{(i-1)} \cup \{\mathbf{s}\} \cup OR^{(i-1)}) = n$. Thus, the following conditions must be also true:

$$\text{rank}(W^{(i-1)} \cup OR^{(i-1)}) = n - 1, \mathbf{s} \perp W^{(i-1)} \cup OR^{(i-1)},$$

and $\mathbf{d} \propto W^{(i-1)} \cup OR^{(i-1)}$. By the algorithm, we know that

$$\text{TempS} = \text{ids}(OR^{(i-1)}, W^{(i-1)}) \text{ and } \mathbf{r} = \text{cv}(\text{TempS}, W^{(i-1)}, \mathbf{d}).$$

We replace vector $\mathbf{r}$ with $\mathbf{r} + \mathbf{s}$ and

$$OR^{(i)} = (OR^{(i-1)} - \{\mathbf{r}\}) \cup \{\mathbf{r} + \mathbf{s}\}.$$

From Proposition 2.3 ii), $\mathbf{d} \perp W^{(i-1)} \cup OR^{(i)} = W^{(i)} \cup OR^{(i)}$. Since $W_i^{(i)} = W^{(i)} \cup \{\mathbf{d}\}$,

$$\text{rank}\left(W_i^{(i)} \cup OR^{(i)}\right) = \text{rank}\left(W^{(i-1)} \cup OR^{(i-1)}\right) + 1 = n.$$

Because $OR^{(i-1)} \ne OR^{(i)}$, we have to verify that

$$\text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) = n \text{ for } j \le i - 1.$$

According to the inductive assumption,

$$\text{rank}\left(W_j^{(i)}\right) \cup OR^{(i-1)} = n \text{ for } j \le i - 1.$$

Let $OR_j^{(i-1)} = \text{ids}\left(OR^{(i-1)}, W_j^{(i)}\right) (j \le i - 1)$, then $OR_j^{(i-1)} \perp W_j^{(i)}$ and $\left|OR_j^{(i-1)}\right| + \text{rank}\left(W_j^{(i)}\right) = n$. If $\mathbf{r} \notin OR_j^{(i-1)}$, then

$$OR_j^{(i-1)} = OR_j^{(i)} \text{ and } \text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) = n.$$

Otherwise, $OR_j^{(i)} - \left(OR_j^{(i-1)} - \{\mathbf{r}\}\right) \cup \{\mathbf{r} + \mathbf{s}\}$. Since $\mathbf{s} \in W_j^{(i)}$, by Proposition 2.5 i),

$$OR_j^{(i)} \perp W_j^{(i)} \text{ and } \left|OR_j^{(i-1)}\right| = \left|OR_i^{(i)}\right|.$$

Thus,

$$\begin{aligned} &\text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) \\ &= \left|OR_j^{(i)}\right| + \text{rank}\left(W_j^{(i)}\right) \\ &= \left|OR_j^{(i-1)}\right| + \text{rank}\left(W_j^{(i)}\right) = n. \end{aligned}$$

CASE 2. According to the algorithm, we consider two subcases:

CASE 2.1. $\mathbf{d} \perp W'$. In this case, we will update X to make $\mathbf{d} \perp W' \cup X^{(i)}$. From the algorithm, $\mathbf{d} = \mathbf{x} + \mathbf{y}$ ($\mathbf{y} \propto W' \cup (X^{(i-1)} - \{\mathbf{x}\})$) and $X^{(i)} = (X^{(i-1)} - \{\mathbf{x}\}) \cup \{\mathbf{x} + \mathbf{s}\}$. Since $\mathbf{s} \perp X^{(i-1)} \cup W'$ (which is the condition of Case 2), $\mathbf{d} \perp X^{(i)} \cup W'$ by Proposition 2.3 ii). Thus, $X^{(i)} \perp W_i^{(i)} = W' \cup \{\mathbf{d}\}$.

On the other hand, $OR^{(i)} = OR^{(i-1)}$. Because of the linear equivalence between W_i^i and $W_{i-1}^{(i-1)}$, $OR_{i-1}^{(i-1)} \perp W_i^{(i)}$ (by Proposition 2.2 iii)). Since $\text{rank}\left(W_j^{(i)}\right) = \text{rank}\left(W_{i-1}^{(i-1)}\right)$, $\text{rank}\left(W_i^{(i)} \cup OR^{(i)}\right) = \text{rank}\left(W_{i-1}^{(i-1)} \cup OR^{(i-1)}\right) = n$.

Since $X^{(i)} = (X^{(i-1)} - \{\mathbf{x}\}) \cup \{\mathbf{x}+\mathbf{s}\} \ne X^{(i-1)}$, we need to verify Cond(I)[i, j] for $j \le i - 1$. Given a $j \le i - 1$, if $\mathbf{s} \in W_j^{(i-1)}$, then $X_j^{(i)} \perp W'_i$ assured by Proposition 2.5 ii). Otherwise, $\mathbf{x} \notin W_j^{(i-1)}$ (because $\mathbf{x}$ must occur later than $\mathbf{s}$ in the extended transition sequence) and $X_j^{(i)} = X_j^{(i-1)}$. Thus, we have $X_j^{(i)} \perp W'_j$ for any $j \le i - 1$. Note that $W_j^{(i)}$ is also linearly equivalent to $W_j^{(i-1)}$. By a similar argument as above,

$$\text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) = \text{rank}\left(W_j^{(i-1)} \cup OR^{(i-1)}\right) = n.$$

CASE 2.2. $\mathbf{d} \propto W'$. Here, $X^{(i)} = X^{(i-1)}$, $W_i^{(i)} = W \cup \{\mathbf{d}\}$. $X_j^{(i)} \perp W'_j$ for $j \le i - 1$ is proved by the inductive assumption and $X_i^{(i)} \perp W'_i$ is proved by Proposition 2.2 i). Let us prove $\text{rank}\left(W_j^{(i)} \cup OR^{(i)}\right) = n$ for $j \le i$. From the discussion in Case 1, we assume $\text{rank}(W^{(i-1)} \cup OR^{(i-1)}) < n$ which implies the following

conditions: $\text{rank}(W^{(i-1)} \cup OR^{(i-1)}) = n-1$, $\mathbf{s} \perp W^{(i-1)} \cup OR^{(i-1)}$, and $\mathbf{d} \propto W^{(i-1)}$.

Let TempS = $\text{ids}(OR^{(i-1)}, W^{(i-1)})$. Since

$$\text{rank}(W^{(i-1)} \cup OR^{(i-1)}) = |\text{TempS}| + \text{rank}(W^{(i-1)}) = n-1$$

and

$\text{rank}(W^{(i-1)}) \geq d_{\min} + k$, $|\text{TempS}| \leq t-1$, i.e., $|OR^{(i-1)} - \text{TempS}| > 0$.

There exists an $\mathbf{r} \in OR^{(i-1)} - \text{TempS}$ and $\mathbf{r} \propto W^{(i-1)} \cup OR^{(i-1)}$ (by the maximality of TempS). Thus,

$$\mathbf{r}+\mathbf{s} \perp W^{(i-1)} \cup OR^{(i-1)}, \text{ and } \text{TempS} \cup \{\mathbf{r}+\mathbf{s}\} \perp W^{(i-1)}.$$

Since

$$|\text{TempS} \cup \{\mathbf{r}+\mathbf{s}\}| = |\text{TempS}| + 1,$$
$$\text{rank}(W^{(i-1)} \cup \text{TempS} \cup \{\mathbf{r}+\mathbf{s}\}) = n,$$
$$\text{rank}(W^{(i)}_j) = \text{rank}(W^{(i-1)}_j),$$

and

$$OR^{(i)} = (OR^{(i-1)} - \{\mathbf{r}\}) \cup \{\mathbf{r}+\mathbf{s}\},$$

it is easy to see that $\text{rank}(OR^{(i)} \cup W^{(i)}_j) = n$. The relations

$$\text{rank}(W^{(i)}_j \cup OR^{(i)}) = n \text{ for } j \leq i-1$$

also can be proved by a similar argument to that in Case 1 and the detail is omitted here.

CASE 3 AND CASE 4. These can be dealt with by similar techniques introduced in Case 1 and Case 2. We omit the detail for simplicity. □

The time complexity of our algorithm is analyzed as the following. Let T_A be the time complexity of procedure Assignment. The total time complexity of the whole algorithm is $T_A + O(N \log N)$, where $O(N \log N)$ is the time for partitioning N source-destination pairs into 2^t groups. In order to estimate the order of T_A, we note that, because $S \cup \{1\}$ is a base of $\mathcal{G}'_n$, each vector in $\mathcal{G}'_n$ can be represented by its coordinate under base $S \cup \{1\}$ which is an $(n+1)$ component vector. Thus, the Assignment can be readily implemented by using the coordinate of each vector instead of using each vector itself. In the Assignment, we have to deal with the total of $n + k + 1 = O(n)$ windows one by one. In each window, there are three important operations which dominate the time complexity:

1) checking whether $\mathbf{d} \perp U$ or not ($|U| \leq n$);
2) computing $\text{cv}(U, V, \mathbf{d})$ ($|U|, |V| \leq n$), where $\mathbf{d} \in \mathcal{G}_n$ and $U, V \subseteq \mathcal{G}_n$; and
3) computing $\text{ids}(U,V)$. With the coordinate of each vector, operations 1) and 2) are both equivalent to solving a system of $O(n)$ linear equations with $O(n)$ variables.

Operation 3) can be done by finding an arbitrary subset of U which is linearly independent of V, then expanding this subset by adding other vectors in U until no other vector can be added. The implementation of this idea needs to solve $O(n)$ systems of linear equations. Therefore, the time complexity of Assignment, T_A, is bounded by $O(n^{O(1)}) = O(\log^{O(1)} N)$. (Note that T_A can be reduced to $O(\log^2 N)$ for BPC permutations where $\mathbf{d} \in S'$.) Thus, the total time complexity is $T_A + O(N \log N) = O(\log^{O(1)} N) + O(N \log N) = O(N \log N)$. Shen [10] indicated that $\Omega(N \log N)$ is the lower bound of the time required to solve a PA problem on an MCTN; we claim that our algorithm is also optimal in time complexity. In summary, our main result is the following theorem.

THEOREM 4.1. *The MP problem for any LC permutation on a k-EMCTN can be solved within O(NlogN) time.*

In the above discussion, we assume that the permutation to be realized is an LC class permutation and the coordinates of each D_i under base $S \cup \{1\}$ are given. An additional but interesting problem is to determine if a given permutation is an LC permutation and how to find the coordinate of each D_i under base $S \cup \{1\}$. Let $\pi = (S_0, S_1, \ldots, S_{n-1}, D_0, D_1, \ldots, D_{n-1})$. If π is an LC class permutation, for any $i = 0, 1, \ldots, n-1$, $D_i = \sum_{j=0}^{n-1} \alpha_{j,i} S_j + \beta_i 1$. Let $A = (\alpha_{j,i})_{n \times n}$, $B = (\beta_i)_{1 \times n}$, $m(S) = (S_0, S_1, \ldots, S_{n-1})_{j \times n}$, and $m(D) = (D_0, D_1, \ldots, D_{n-1})_{j \times n}$. We have the following equation:

$$m(D) = m(S) \times A + 1 \times B. \tag{4-1}$$

Note that $D_i = \sum_{j=0}^{n-1} \alpha_{j,i} S_j + \beta_i 1$ consists of N equations such that

$$d_{j,i} = \alpha_{0,i}\, s_{j,0} \oplus \alpha_{1,i}\, s_{j,1} \oplus \ldots \oplus \alpha_{n-1,i}\, s_{j,n-1} \oplus \beta_i 1, \qquad (j = 0, 1, \ldots, N-1).$$

There are N equations and $(n+1)$ variables $(\alpha_{0,i}, \alpha_{1,i}, \ldots, \alpha_{n-1,i}, \beta_i)$. From the *1st*, *2nd*, *3rd*, *5th*, *9th*, ..., $(2^{n-1}+1)$st equation, we have

$$\beta_i = d_{0,i}\,; \text{ and } \alpha_{j,i} = d^{2(n-1)-j,i} \oplus \beta_i. \text{ for } j = 0, 1, \ldots, n-1. \tag{4-2}$$

Therefore, the algorithm is quite simple. We use (4-2) to obtain the matrix A and vector B first, then check whether they satisfy (4-1). If (4-1) is satisfied, then the permutation π is an LC class permutation and A and B give out the coordinates of each D_i under base $S \cup \{1\}$, otherwise, π is NOT an LC permutation. Since obtaining A and B needs $O(n^2) = O(\log^2 N)$ time and computing (4-1) can be done by the fast matrix multiplication [19] within $O(Nn^{1.376})$ time, the time complexity of this algorithm is bounded by $O(Nn^{1.376}) = O(N \log^{1.376} N)$.

5 CONCLUSION

In this paper, we have proposed an algorithm for realizing any LC permutation on a k-Ω (as well as k-EMCTN) with the minimum number of passes within the minimum time. Thus, we have solved the MP problem for LC permutations on k-EMCTN. Compared with the existent results[12], [13], this algorithm can be applied to many more permutations. Besides, our algorithm is simpler and clearer than that of [12], [13]. Moreover, the total time (including overhead) of applying this algorithm to BPC permutations remains the same as those in [12], [13].

ACKNOWLEDGMENTS

The authors thank Stephanie Griffin for helpful comments on the writing. This work was supported in part by the Office of UMKC Research Administration under Grant No. K-2-11134.

REFERENCES

[1] C. Wu and T. Feng, "The universality of the shuffle-exchange network," *IEEE Trans. Computers*, vol. 30, pp. 324-332, May 1981.
[2] D.P. Agrawal, "Graph theoretical analysis and design of multistage interconnection networks," *IEEE Trans. Computers*, vol. 32, pp. 637-648, July 1983.
[3] C. Wu and T. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Computers*, vol. 29, pp. 694-702, Aug. 1980.
[4] G.B. Adams III and H.J. Siegel, "The extra stage cube: a fault-tolerant interconnection network for supersystems," *IEEE Trans. Computers*, vol. 31, pp. 443-454, May 1982.
[5] C.L. Wu, T.Y. Feng, and M.C. Lin, "Star: A local network system for real-time management of imagery data," *IEEE Trans. Computers*, vol. 31, pp. 923-933, Oct. 1982.
[6] I. Gazit and M. Malek, "On the number of permutations performable by extra-stage multistage interconnection networks," *IEEE Trans. Computers*, vol. 38, no. 2, pp. 297-302, Feb. 1989.
[7] C.-T. Lea and D.-J. Shyy, "Tradeoff of horizontal decomposition versus vertical stacking in rearrangeable nonblocking networks," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 899-904, June 1991.

[8] D.-J. Shyy and C.-T. Lea, "$Log_2(N, m, p)$ strictly nonblocking networks," *IEEE Trans. Computers*, vol. 39, no. 10, pp. 1502-1510, Oct. 1991.
[9] X. Shen, M. Xu and X. Wang, "An optimal algorithm for permutation admissibility to multistage interconnection networks," *IEEE Trans. Computers*, vol. 44, no. 4, pp. 604-608, Apr. 1995.
[10] X. Shen, "An optimal O(NlgN) algorithm for permutation admissibility to extra-stage cube-type networks," *IEEE Trans. Computers*, vol. 44, no. 9, pp. 1,144-1,149, Sept. 1995.
[11] C.J.A. Hsia and C.Y. Chen, "Permutation capability of multistage interconnection networks," *Proc. Int'l Conf. Parallel Processing*, 1990, pp. I-338-346.
[12] C.S. Raghavendra and A. Varma, "Fault-tolerant multiprocessors with redundant-path interconnection networks," *IEEE Trans. Computers*, vol. 35, no. 4, pp. 307-316, Apr. 1986.
[13] X. Shen, "Optimal realization of any BPC permutation on k-extra-etage cube-type networks," *IEEE Trans. Computers*, vol. 44, no. 5, pp. 714-719, May 1995..
[14] A. Sengupta, K. Zemoudeh, and S. Bandyopadhyay, "Self-routing algorithms for strongly regular multistage interconnection networks," *J. Parallel and Distributed Computing*, vol. 14, pp. 187-192, 1992.
[15] C.S. Raghavendra and R.V. Boppana, "On self-routing in Benes and shuffle-exchange networks," *IEEE Trans. Computers*, vol. 40, no. 9, pp.1057-1064, Sept. 1991.
[16] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.
[17] H.J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*, Second Edition, McGraw-Hill, 1990.
[18] K. Hoffman and R. Kunze, *linear algebra*, Englewood Cliffs, NJ: Prentice Hall, 1971.
[19] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Proc. 19th Ann. Symp. Theory of Computing*, pp. 1-6, 1987.

On the Analysis and Design of Group Theoretical *t*-syEC/AUED Codes

Chi-Sung Laih, *Member, IEEE*, and Ching-Nung Yang

Abstract—An efficient algorithm to count the cardinalities of certain subsets of constant weight binary vectors is presented in this paper. The algorithm enables us to design 1-symmetric error correcting/all unidirectional error detecting (1-syEC/AUED) codes with the highest cardinality based on the group Z_n. Since a *field* Z_p is a group, this algorithm can also be used to design a *field* 1-syEC/AUED code. We can construct t-syEC/AUED codes for $t = 2$ or 3 by appending a tail to the *field* 1-syEC/AUED codes. The information rates of the proposed t-syEC/AUED codes are shown to be better than the previously developed codes.

Index Terms—Balanced codes, error correction, error detection, unidirectional errors, combinatorics, partition

1 Introduction

ERROR control codes have been widely used in the communication and computer storage systems to enhance their reliability and integrity [1], [2], [3]. It is observed that many faults in VLSI memories, PLA, shift registers are due to unidirectional errors [4]. Both kinds of unidirectional errors, i.e., $0 \rightarrow 1$ and $1 \rightarrow 0$, could occur in a message communication. However, they do not occur in the same codeword. The first published work in this area is by Pradhan [4]. The t-syEC/AUED codes that can correct t or fewer symmetric errors and detect all unidirectional errors [6], [7], [8], [9], [10], [11], [12] are the extensions of 1-syEC/AUED codes. In [5], using a group theoretical method, Bose and Rao proposed a novel method of designing 1-syEC/AUED codes for any given length n which can be described as follows:

Consider a group $Z_n = \{0, 1, \ldots, n-1\}$. Let $B = \{0, 1\}$, and consider the set $V^{(w)} = \{v \mid v \in B^n$ where v has w 1s and $n - w$ 0s$\}$, i.e., $V^{(w)}$ is the set of binary n-tuple with constant weight w. Define a function $T(\cdot)$ as follows:

$$T(v) = T\left((a_0, a_1, \ldots, a_{n-1})\right) = \sum_{i=0}^{n-1} a_i i.$$

The sum operation of the above function is in Z_n, i.e., mod n. Then a code C_j can be constructed by the function $T(\cdot)$ as follows:

$$C_j = \left\{v = (a_0, a_1, \ldots, a_{n-1}) \in V^{(w)} \,\middle|\, T(v) = j\right\}.$$

The Hamming distance between two codewords of C_j is clearly even because C_j is a constant weight code. If the Hamming distance is 2 between two codewords, then it is easy to show that they cannot both be in C_j. Therefore, the Bose-Rao codes have minimum distance 4 between codewords. As shown in [5], C_j is 1-syEC/AUED code.

We wish to find out which C_j has the highest cardinality for obtaining 1-syEC/AUED codes with as high code rate as possible. Since the method proposed by Bose and Rao is based on the exhaustive search to find C_j, how to effectively count C_j is a challenge.

• *The authors are with the Communication Laboratory, Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, Republic of China. E-mail: laihcs@eembox.ncku.edu.tw.*

Manuscript received Dec. 9, 1993; revised Nov. 1, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95120.