國立交通大學

資訊科學與工程研究所

碩 士 論 文

排 列 碼 與 其 高 效 率 之 編 解 碼 演 算 法

Efficient Encoding and Decoding with Permutation Arrays

研 究 生：林德璁

指導教授：蔡錫鈞 教授

中 華 民 國 九 十 六 年 七 月

排 列 碼 與 其 高 效 率 之 編 解 碼 演 算 法
Efficient Encoding and Decoding with Permutation Arrays

研　究　生：林德璁　　　　Student：Te-Tsung Lin

指導教授：蔡錫鈞　　　　Advisor：Shi-Chun Tsai

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

Efficient Encoding and Decoding with Permutation Arrays

Te-Tsung Lin

Abstract

An (n, d) permutation array(PA) is a subset of S_n with the property that the distance (under any distance metric, such as hamming) between any two permutations in the array is at least d, which becomes popular recently for communication over power line. We use both hamming distance and l_∞-norm to measure the distance between permutations, and give constructions of permutations arrays under those two metrics. For the hamming distance, we give the first explicit construction of 3-DPM$_H$. For the l_∞-norm, we give the first explicit construction of DPM$_\infty$ and a direct construction of (n, d) permutation array with l_∞-norm without using other binary code. Furthermore, all have efficient encoding and decoding algorithms.

Acknowledgements

I am grateful to my advisor, Dr. Shi-Chun Tsai, for his guidance. I thank Hsin-Lung Wu who gives some important ideas for this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Preliminary

Let S_n denote the set of all permutations of length n. We consider a coding scheme $C : \{0,1\}^k \rightarrow S_n$ with the property that if we are given a permutation $y \in S_n$ that is close to a valid encoding $C(x)$, then it is possible to recover the message x from the corrupted encoding y. To do this, first we need to choose a proper metric for the distance between two permutations. A natural choice is hamming distance, but it is not clear how to decode corrupted permutations efficiently under this metric. Secondly, we need to decide which permutations can be used as code words, such that for any two different messages x and x', such that $C(x)$ and $C(x')$ are "far" enough. In this thesis, we give efficient encoding and decoding algorithms for such scheme by measuring the permutation distance with the hamming distance and the l_∞-norm.

An (n, d) permutation array(PA) is a subset of S_n with the property that the distance (under any distance metric, such as hamming, etc.) between any two permutations in the array is at least d. PAs were studied for some time [6]. It is Vinck [25], who proposed permutation arrays as an error correcting code over power-line communications, where each symbol $i \in \{1, .., n\}$ is associated with a frequency f_i and a message is encoded as a permutation, which is then transmitted in time as the series of corresponding frequencies. For example, to transmit the message encoded as $(3, 4, 1, 2)$, the sequence of frequencies (f_3, f_4, f_1, f_2) is transmitted one by one. Since then many researches have been done on coding/modulation schemes with PAs [12],[21],[23],[24]. Ferreira and Vinck [12] made use of distance preserving mappings (DPMs) from binary sequences to permutation sequences to construct permutation trellis codes. A systematic study of DPMs was initiated in [6]. Later, Lee [16],[18], Swart et al. [20] proposed several constructions of DPMs, and Chang [4],[5] studied the distance increasing mappings (DIMs). All the above mentioned works use the hamming distance as a metric for permutations. Most of the efforts have been on finding mappings from binary vectors to permutations that preserve the minimum distance of the binary vectors. A typical scheme is starting by encoding a message with a binary code, which is mapped to a permutation and transmitted. Upon receiving a permutation, one can recover the corresponding binary vector and then with the binary code one can do some error correcting to recover the message. However, there is no discussion on the efficiency of error correcting directly from the permutations. In our research, we construct distance preserving mappings from ternary vectors to permutations and we give efficient encoding/decoding scheme for the PAs we constructed.

A permutation can be seen as a ranking, and vice versa. To study the correlations between ranks, several metrics on permutations were introduced, such as the hamming distance, the minimum number of transpositions taking one permutation to another, etc. [14], [9], [7], [8]. And some consider permutation arrays with different metrics. Stoll and Kurz [22] investigated a detection scheme of permutation arrays using Spearman's rank correlation. Chadwick and Kurz [3] studied the permutation arrays based on Kendall's tau.

We consider a noisy channel which can transmit permutations as code words. The noise in the channel is an independent Gaussian distribution with zero mean for each position. The received sequence is the original permutation together with the Gaussian noise, and its ranking can be seen as a permutation, which can be different from the original one. Under the model of additive white gaussian noise (AWGN) [11], there is only a small probability for any frequency to deviate significantly from the original one. This inspires us to consider not only the hamming distance but also the l_∞-norm.

1.2 Main Result and Construction Idea

In this thesis, we have two main results.

First, we give the first explicit construction of distance preserving mappings from ternary vectors of dimension n to S_n with hamming distance (3-DPM$_H$) for $n \geq 16$. Thus we can construct (n, d) permutation array under hamming distance with size $\geq A_3(n, d)$. Moreover, we have efficient encoding/decoding scheme for the PAs we constructed by the 3-DPM$_H$.

Second, we give explicit constructions of distance preserving mappings with l_∞-norm (DPM$_\infty$), which can be used to recover corrupted permutations. And we give an (n, d) permutation array under l_∞-norm without using binary codes. It's the first direct construction of PAs to the best of our knowledge. With both constructions, a lower bound on the size of permutation arrays is given, i.e. $P_\infty(n, d) \geq A(n-1, d)$, and $P_\infty(n, d) \geq 2^{n-d}$. Moreover, for both constructions, we have efficient encoding/decoding algorithms.

For the first result, Our 3-DPM$_H$ construction is inspired by [18]. It is based on a crucial "local" property which we discuss as follows. Intuitively, an algorithm has the local property if each element of the permutation is not far away from its initial position after running the algorithm. From a 2-DPM$_H$ with local property, we can obtain a 3-DPM$_H$. First we run a 2-DPM$_H$ algorithm such that every element in the permutation is not far from the initial position, i.e. with a small position difference. Then we only swap two positions far enough, i.e. with the position difference larger than the difference resulting from the 2-DPM$_H$. This will give us a 3-DPM$_H$ if we have a 2-DPM$_H$ with local property. We constructed a two-pass 3-DPM$_H$ by using a 2-DPM$_H$, which is very similar to the one constructed in [17, 18]. However, in these papers, the local property is not fully exploited. Following the same paradigm, one can obtain q-DPM$_H$ for all $q > 3$.

For the second result, both construction ideas are crucial on a greedy strategy. For a binary vector, first we use the largest number n to represent 1 and the smallest number to represent 0 for the first bit. For the second bit, we use the available largest number to represent 1 and the available smallest number to represent 0, i.e. if the first and second bit is one, then we use the largest value n to represent the first bit and second largest value $n-1$ to represent the second bit. The other bits can be determined one by one. Thus each value of permutation only depends on the prefix of the vector, and then it gives the largest distance with a greedy strategy and it can be decoded in linear time.

1.3 Notations

Let $[n] = \{1, \cdots, n\}$, $[m \cdots n] = \{m, m+1, \cdots, n\}$, for $m < n$. For a function f, let $f(S)$ denote the union of $f(s)$ for all $s \in S$. Let $\delta : Z_q \times Z_q \to \{0, 1\}$ be the function defined by $\delta(a, b) = 1$ if $a \neq b$ and 0 otherwise. Let S_n denote the set of all permutations of $[n]$ and Z_q^n denote the set of all q-ary vectors of length n. For any $\pi \in S_n$ and $i \in [n]$, $\pi^{-1}(i)$ denotes the position of i in π, i.e. if $\pi(j) = i$ then $\pi^{-1}(i) = j$. Let id_n denote the identity permutation in S_n, i.e. $id_n = (1, 2, \cdots, n)$. For any $x \in Z_2^n$, we use $x_{[i..j]}$ to denote the subvector $(x_i, \cdots, x_j)$ for any $i < j$. For any $\pi \in S_n$, we use $\pi_{[i..j]}$ to denote the partial permutation $(\pi_i, \cdots, \pi_j)$ for any $i < j$. The Hamming distance $d_H(a, b)$ between two n-tuples $a = (a_1, a_2, \cdots, a_n)$ and $b = (b_1, b_2, \cdots, b_n)$ is the number of positions where they differ, i.e. $d_H(a, b) = |\{j : a_j \neq b_j\}|$. The l_∞-norm distance of two permutations is $d_\infty(\pi, \sigma) = \max_j |\pi_j - \sigma_j|$.

Define $V_f(n, d) = |\{\pi \in S_n : d_f(id, \pi) \leq d\}|$ to be the size of a sphere with center $id \in S_n$ and radius d, where f is any metric function of S_n. If f is right-invariant, i.e., $d_f(\pi_1, \pi_2) = d_f(\pi_1\sigma, \pi_2\sigma)$ for all σ then for any center π and fixed radius d, the size of a sphere is the same, i.e. $|\{\pi \in S_n : d_f(\sigma, \pi) \leq d\}| = \{\pi \in S_n : d_f(id, \pi\sigma^{-1}) \leq d\}| = V_f(n, d)$. Let (n, d) q-ary code be a code over Z_q^n with minimum distance d. Let (n, d)-PA with metric f be a permutation array over S_n with minimum distance d based on metric f. Let $A_q(n, d)$ denote the maximum size among all (n, d) q-ary code and $P_f(n, d)$ denote the maximum size among all (n, d)-PA with metric f. A mapping $F : Z_q^{n_1} \to S_{n_2}$ is a q-ary distance-preserving mappings under metric f (q-DPM$_f$), if for any $x, y \in Z_2^{n_1}$, $d_f(F(x), F(y)) \geq d_H(x, y)$. We usually omit q for $q = 2$.

Chapter 2

Metrics and Lower Bound of Permutation Arrays

In this chapter, we introduce several metrics, and derive the Gilbert like lower bounds for permutation arrays under these metrics. To get the lower bounds, we will need to estimate the size of a sphere for every metric.

2.1 Metrics on S_n

Given S_n and a metric function $d : S_n \times S_n \to R^+$ satisfied $d(\pi, \pi) = 0$, $d(\pi, \sigma) = d(\sigma, \pi)$ and $d(\pi, \sigma) \leq d(\pi, \eta) + d(\eta, \sigma)$ then (S_n, d) formed a metric space. The metric function d is designed to measure the distance between any two permutations in S_n. We call the metric function just metric.

Many metrics can be defined and discussed. We need two additional restrictions. First is right invariant. In general, permutations are presented as one to one mappings between two sets with the same cardinality. $\pi : A \to B$, $|A| = |B| = n$. If the distance will not change when changing the labeling of A, then it's right invariant, i.e. $d(\pi_1, \pi_2) = d(\pi_1\sigma, \pi_2\sigma)$ for all

σ. On the other hand, if the distance will not change when changing the labeling of B, it's left invariant, $d(\pi_1, \pi_2) = d(\sigma\pi_1, \sigma\pi_2)$. By the definition, given a right invariant metric d, it's easy to construct another inverse metric $d'(\pi_1, \pi_2) = d(\pi_1^{-1}, \pi_2^{-1})$ which is left invariant. It's because $d'(\pi_1, \pi_2) = d(\pi_1^{-1}, \pi_2^{-1}) = d(\pi_1^{-1}\sigma, \pi_2^{-1}\sigma) = d((\sigma^{-1}\pi_1)^{-1}, (\sigma^{-1}\pi_2)^{-1}) = d'(\sigma^{-1}\pi_1, \sigma^{-1}\pi_2)$. So we only need to consider right invariant.

Next we introduce several different kinds of metrics. These metrics have been used to measure the distance of permutations in various areas.

Define $V_f(n, d) = |\{\pi \in S_n : d_f(id, \pi) \leq d\}|$, the size of a sphere with center $id \in S_n$ with radius d with respect to metric f. Note that all metrics we discuss here are right-invariant, so spheres with the same radius have the same sizes, i.e., given any $\sigma \in S_n$, $|\{\pi \in S_n : d_f(\sigma, \pi) \leq d\}| = \{\pi \in S_n : d_f(id, \pi\sigma^{-1}) \leq d\}| = V_f(n, d)$.

The Hamming distance is a well-known and very popular metric. Originally it's a natural design for string. It counts the number of positions for which the corresponding symbols are different. Hamming distance is widely used for binary vectors and q-ary vectors in coding theory category. The Hamming distance between two permutations is $d_H(\pi, \sigma) = |\{j : \pi_j \neq \sigma_j\}|$. One may verify that it's a bi-invariant metric easily. Next let's consider $V_{d_H}(n, d)$. Because $|\{\pi | d_H(id, \pi) = d\}| = \binom{n}{d}(!d)$, where the subfactorial $!d$ is the number of distinct derangement on d elements, it implies $V_{d_H}(n, d) = \sum_{i=0}^{d} \binom{n}{i}(!i)$. It's well known the subfactorials satisfy the recurrence relations $!(n+1) = n \cdot [!n + !(n-1)]$ and $!n$ is equivalent to $ning(\frac{n!}{e})$, where $ning$ is the nearest integer function, $ning(r) = minarg_j\{z \in Z : |j - r|\}$(half-integers are rounded to even numbers to avoid ambiguous). Thus $V_{d_H}(n, d) \leq \frac{2}{e}\frac{n!}{(n-d)!}$ for $d < n$ by $\binom{n}{i-1}(!(i-1)) \leq \frac{1}{2}\binom{n}{i}(!i)$ if $d < n$.

There is another famous norm called l_1-norm, and it's defined as $l_1(\pi,\sigma) = \sum_{j=1}^{n} |\pi_j - \sigma_j|$. It's not clear whether an explicit formula of $V_{l_1}(n,d)$ exist, but the upper bound can be derived. By [10], $V_{l_1}(n,d) \le (\frac{2e(d+n)}{n})^n$. Note that l_1-norm is one of a metric family called l_p-norm family. In general, l_p-norm is defined as $l_p(\pi,\sigma) = [\sum_{j=1}^{n}(|\pi_j - \sigma_j|)^p]^{\frac{1}{p}}$.

The l_∞-norm of two permutations is $d_\infty(\pi,\sigma) = \max_j |\pi_j - \sigma_j|$. It's a special case of l_p-norm when p is infinitely large. We give two ways to derive upper bounds for $V_\infty(n,d)$. Note that there is a connection between $V_\infty(n,d)$ and permanent. Recall the definition of permanent for a matrix A, $perA \equiv \sum_{\pi\in S_n} a_{1\pi_1}\cdots a_{n\pi_n} = |\{\pi \in S_n : a_{i\pi_i} = 1 \text{ for all } i\}|$. Define $A^{(n,d)}$ to be an $n\times n$ matrix, $a_{ij}^{(n,d)} = \begin{cases} 1 & \text{if } |j-i| \le d \\ 0 & otherwise \end{cases}$

And by the theorem 11.5 in [19], for an $n \times n$ $(0,1)$-matrix A with r_i ones in row i, then $per(A) \le \sum_{i=1}^{n}(r_i)!^{\frac{1}{r_i}}$. We can derive the upper bound of $V_\infty(n,d)$ as follows.

$$\begin{aligned} V_\infty(n,d) &= |\{\pi \in S_n : d_\infty(id,\pi) \le d\}| \\ &= |\{\pi \in S_n : |\pi_i - i| \le d \text{ for all } i\}| \\ &= |\{\pi \in S_n : a_{i\pi_i}^{(n,d)} = 1 \text{ for all } i\}| \\ &= per(A^{(n,d)}) \\ &\le [(2d+1)!]^{\frac{n}{2d+1}} \end{aligned}$$

The second way to estimate $V_\infty(n,d)$ is by a recurrence relation. Define A_{ij} to be the matrix obtained from A by deleting row i and column i. It's well known $per(A) = \sum_{i=1}^{n} a_{ij}\cdot per(A_{ij})$. By observing $A^{(n,d)}$, one can find $A_{11}^{(n,d)} = A^{(n-1,d)}$ and each entry in $A_{1k}^{(n,d)}$ is upper bounded by the corresponding entry

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Figure 2.1: $A_{6\times 6}$ such that $per(A) = V_\infty(6,1)$

of $A^{(n-1,d)}$ for $2 \leq k \leq 1+d$. Thus

$$\begin{aligned} V_\infty(n,d) &= per(A^{(n,d)}) \\ &= \sum_{k=1}^{1+d} per(A_{1k}^{(n,d)}) \\ &\leq (1+d)per(A^{(n-1,d)}) \\ &\leq (1+d)^2 per(A^{(n-2,d)}) \\ &\cdots \\ &\leq (1+d)^{n-d-1} per(A^{(d+1,d)}) \\ &= (1+d)^{n-d-1} V_\infty(d+1,d) \\ &= (1+d)^{n-d-1}(d+1)! \end{aligned}$$

The second bound is better when d is large.

For other metrics, define $I(\pi,\sigma)$ as the minimum number of pairwise adjacent transpositions taking π^{-1} to σ^{-1}. It has an equivalent definition $I(\pi,\sigma) \equiv |\{(i,j) : \pi_i < \pi_j, \sigma_i > \sigma_j\}|$. Let π be a permutation $\in S_n$. If $i < j$ and $\pi_i > \pi_j$, the pair (i,j) is called an inversion of π. Note that $I(\pi,\sigma)$ equals the number of inversions of $\pi\sigma^{-1}$. Let $I_n(k)$ denotes the number of permutations $\in S_n$ with exactly k inversions. By [15], $I_n(k)$ have a recurrence relation $I_n(k) = I_{n-1}(k) + I_{n-1}(k-1) + I_{n-1}(k-2) + .. + I_{n-1}(k-n+1)$ and then $V_I(n,d) = |\{\pi \in S_n : I(\pi, id) \leq d\}| = |\{\pi \in S_n :$ the number of inversions of $\pi \leq d\}| = \sum_{k=0}^{d} I_n(k)$. Thus $V_I(n,d)$ can be computed by dynamic programming in quadratic time.

Define $T(\pi, \sigma)$ as the minimum number of transpositions required to bring π to σ. This is a bi-invariant metric on S_n. It's known $T(\pi, \sigma) = n-$ number of cycles in $\pi\sigma^{-1}$ [9]. Let $c(n,k)$ denote the number of permutations $\pi \in S_n$ with exactly k cycles. This number is called a signless Stirling number of the first kind. $c(n,k)$ satisfies the recurrence relation $c(n,k) = (n-1)c(n-1,k) + c(n-1,k-1)$ by [19]. Thus $V_T(n,d) = \sum_{i=0}^{d} c(n, n-i)$, which can be computed by dynamic programming in quadratic time.

2.2 Lower Bound of PAs

Gilbert bound [13] is a lower bound on $A(n,d)$. Similar idea can be applied to permutation arrays.

Theorem 1. $P_f(n,d) \geq \frac{n!}{V_f(n,d-1)}$, *where f is any metric function of S_n.*

Proof. We give a greedy algorithm for producing a permutation array achieving the claimed bound.

(a) Start with any permutation in S_n.

(b) Choose a permutation whose distance is at least d to all previous chosen permutations.

(c) Repeat step (b) as long as there is such a permutation.

Let P be the permutation array produced by the above greedy algorithm. Once the algorithm stops, it implies all permutations can be covered with the $|P|$ spheres centered at codewords in P. Thus $n! \leq |P| \cdot V_d(n, d-1)$

□

By the upper bounds of $V_{d_H}(n,d)$, $V_{l_1}(n,d)$ and $V_\infty(n,d)$, we have following corollaries immediately.

Corollary 1. $P_\infty(n,d) \geq \frac{n!}{[(2d-1)!]^{\frac{n}{2d-1}}}$, *and* $P_\infty(n,d) \geq \frac{n!}{d^{n-d}(d)!}$

Corollary 2. $P_{l_1}(n,d) \geq \frac{n!}{(\frac{2e(d+n)}{n})^n}$

Corollary 3. $P_H(n,d) \geq \frac{n!}{\frac{2}{e}\frac{n!}{(n-d)!}}$ *for* $d < n$.

We give the lower bounds for $n = 16$ with different metrics by following table. One can find that the lower bound of $P_{l_1}(n,d)$ is very large since the l_1-norm has a wide range up to $n^2/2$. By [5], one can construct an (n,d) permutation array P with hamming distance such that $|P| = A(16, d-2)$. Let $U[A(16, d-2)]$ denote the upper bound of $A(16, d-2)$. The lower bound of $P_H(n,d)$ is much larger than $A(16, d-2)$. The large gap between those two constructions inspires us to construct PAs directly without using DPMs/DIMs. Note that the permutation array meets the lower bound of $P_H(n,d)$ by Gilbert bound may not have efficient encoding/decoding algorithm.

$P(16,d)$	$d =$ 3	4	5	6	7	8	9	10	11
l_1	$1549 \cdot 10^8$	$261 \cdot 10^8$	$56 \cdot 10^8$	$1439 \cdot 10^6$	$423 \cdot 10^6$	$139 \cdot 10^6$	$50 \cdot 10^6$	$19 \cdot 10^6$	$8 \cdot 10^6$
T	3122338440	92948453	4082716	250023	20679	2269	327	62	15
l_∞	4647716	72097	3570	480	102	30	12	5	3
$Hamming$	$1729 \cdot 10^8$	$168 \cdot 10^8$	1187378122	99721132	8972294	888754	97568	12013	168
$U[A(16,d-2)]$	65536	32768	3276	2048	340	256	37	32	6

Table 2.1: Lower bounds of $P(16,d)$ with different metrics

Chapter 3

DPMs from Z_3^n to S_n with Hamming Distance

3.1 Construction of 3-DPM$_H$

In this section, we give the construction of **3-DPM**$_H$. First of all, we show the algorithm for input length $8n$ for any integer $n \geq 2$. We call the algorithm A_{8n}. Then we extend A_{8n} for all input length ≥ 16. Note that our approach gives a framework for designing general q-DPM$_H$. In this chapter, all addition and substraction is operated in $Z_{8n} = [8n]$, that is, if $a, b \in Z_{8n}$ then the output of $a + b$ is $a + b \bmod 8n$ if $a + b \bmod 8n \neq 0$, $8n$ otherwise.

3.1.1 3-DPM$_H$ of length $8n$ for $n \geq 2$

The 3-DPM$_H$ of length $8n$ (A_{8n}) is shown in Figure 3.1. Algorithm A_{8n} consists of two passes: PASS 1 and PASS 2. The transition patterns of both passes are illustrated in figures 3.2 and 3.3 respectively.

In figures 2(a) and 3(a), the thin lines represent the transpositions in the first for-loop of both passes and the thick lines represent those transpositions

Algorithm A_{8n}:

Input: $(x_1, \cdots, x_{8n}) \in Z_3^{8n}$

Output: $(\pi_1, \cdots, \pi_{8n}) \in S_{8n}$

PASS 1 :

$(\pi_1^1, \pi_2^1, \cdots, \pi_{8n}^1) \leftarrow (1, 2, \cdots, 8n)$;

for $i = 0$ **to** $4n - 1$ **do**;

  if $x_{2i+1} = 1$ **then** swap $(\pi_{2i+1}^1, \pi_{2i+2}^1)$;

for $i = 0$ **to** $4n - 1$ **do**;

  if $x_{2i+2} = 1$ **then** swap $(\pi_{2i+2}^1, \pi_{2i+3}^1)$;

PASS 2 :

$(\pi_1, \pi_2, \cdots, \pi_{8n}) \leftarrow (\pi_1^1, \pi_2^1, \cdots, \pi_{8n}^1)$;

for $i = 0$ **to** $n - 1$ **do**;

  if $x_{8i+1} = 2$ **then** swap (π_{8i+1}, π_{8i+5});

  if $x_{8i+2} = 2$ **then** swap (π_{8i+2}, π_{8i+6});

  if $x_{8i+3} = 2$ **then** swap (π_{8i+3}, π_{8i+7});

  if $x_{8i+4} = 2$ **then** swap (π_{8i+4}, π_{8i+8});

for $i = 0$ **to** $n - 1$ **do**;

  if $x_{8i+5} = 2$ **then** swap (π_{8i+5}, π_{8i+9});

  if $x_{8i+6} = 2$ **then** swap $(\pi_{8i+6}, \pi_{8i+10})$;

  if $x_{8i+7} = 2$ **then** swap $(\pi_{8i+7}, \pi_{8i+11})$;

  if $x_{8i+8} = 2$ **then** swap $(\pi_{8i+8}, \pi_{8i+12})$;

Output $(\pi_1, \cdots, \pi_{8n})$.

Figure 3.1: 3-DPM$_H$ Algorithm A_{8n}

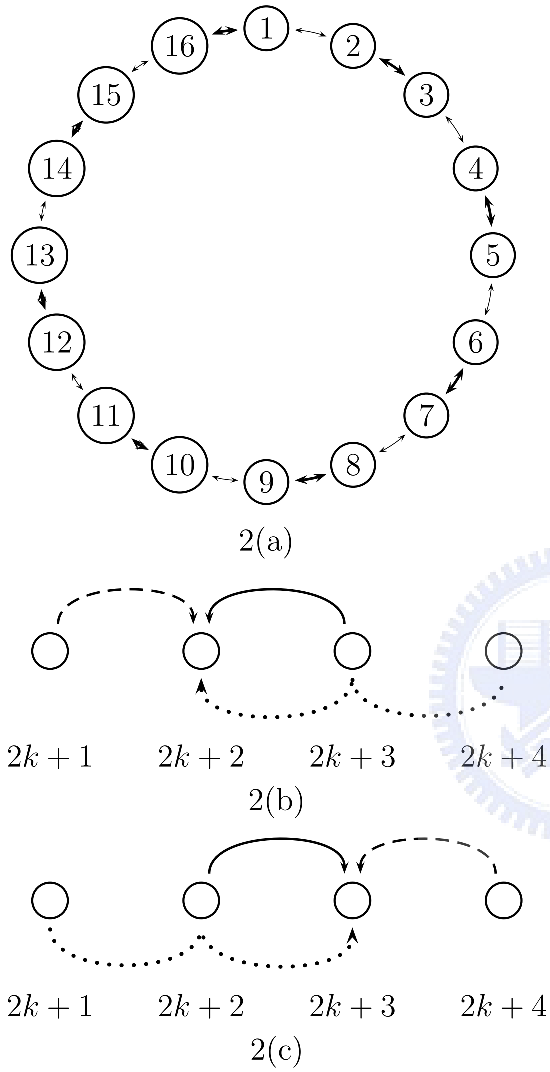

Figure 3.2: Transition patterns of PASS 1.

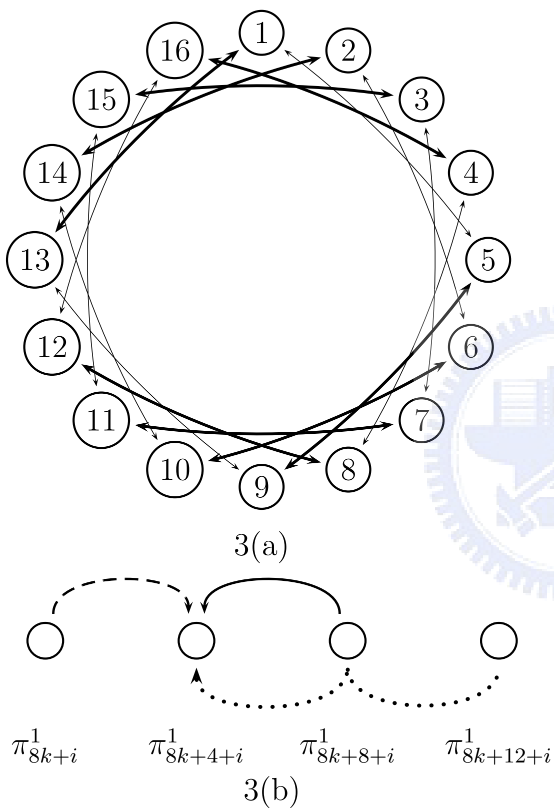

Figure 3.3: Transition patterns of PASS 2. $i \in \{0, 1, 2, 3\}$

in the second for-loop. Note that PASS 1 has the "local" property which is implicitly used in [17, 18]. Since all transpositions in a single for-loop are independent and can be done simultaneously, the local property can be observed in figure 3.2. Now we prove the distance preserving property of A_{8n}.

Theorem 2. *A_{8n} is a 3-DPM$_H$ for all $n \geq 2$.*

Proof. Given $x \in Z_3^{8n}$, let $\pi = A_{8n}(x)$ and π^1 be the intermediate result after PASS 1. First of all, for any fixed position i, we look into what possible values π_i and π_i^1 can be after running the corresponding pass of A_{8n}.

Claim 1. *If i is even, the possible values of π_i^1 are in $\{i-1, i, i+1, i+2\}$. If i is odd, the possible values of π_i^1 are in $\{i-2, i-1, i, i+1\}$. If $i = 8k+4+j$ for $j \in \{0, 1, 2, 3\}$, the possible values of π_i are in $\{\pi_{i-4}^1, \pi_i^1, \pi_{i+4}^1, \pi_{i+8}^1\}$. If $i = 8k + 8 + j$ for $j \in \{0, 1, 2, 3\}$, the possible values of π_i are in $\{\pi_{i-8}^1, \pi_{i-4}^1, \pi_i^1, \pi_{i+4}^1\}$.*

Proof. First consider i is even. Let $i = 2k + 2$. Observe figure 3.2(b), the possible values of π_{2k+2}^1 are $\{2k + 1, 2k + 2, 2k + 3, 2k + 4\}$. For example, if $x_{2k+1} \neq 1$, $x_{2k+2} = 1$ and $x_{2k+3} = 1$ (transition indicated in dotted line), $\pi_{2k+2}^1 = 2k + 4$. If only $x_{2k+2} = 1$(normal line), $\pi_{2k+2}^1 = 2k + 3$. If only $x_{2k+1} = 1$(dashed line), $\pi_{2k+2}^1 = 2k+1$. If all inputs are zero, $\pi_{2k+2}^1 = 2k+2$. Similarly for odd i, the transition pattern is shown in figure 3.2(c). All cases are summarized in Table 3.1.

In the table, each row stands for the input and the corresponding result after swap operations. For example, in row 7, if $x_{2k+1} = x_{2k+2} = 1$ and $x_{2k+3} \neq 1$, then $\pi_{2k+2}^1 = 2k + 3$ and $\pi_{2k+3}^1 = 2k + 1$. Thus by a similar observation from figure 3.3, we summarize the possible values of π_i's in Table 3.2, which is very similar to Table 3.1 if we replace 1 by 2. The claim is true by Table 3.1 and Table 3.2.

	x_{2k+1}	x_{2k+2}	x_{2k+3}	π^1_{2k+2}	π^1_{2k+3}
1	-	-	-	2k+2	2k+3
2	-	-	1	2k+2	2k+4
3	-	1	-	2k+3	2k+2
4	-	1	1	2k+4	2k+2
5	1	-	-	2k+1	2k+3
6	1	-	1	2k+1	2k+4
7	1	1	-	2k+3	2k+1
8	1	1	1	2k+4	2k+1

Table 3.1: Possible values of π^1_j after PASS 1 for $k \in \{0, 1, \cdots, 4n-1\}$.

□

Given $x, y \in \{0,1\}^{8n}$, let $A_{8n}(x) = \pi$, $A_{8n}(y) = \tau$, and π^1 and τ^1 are the intermediate result after PASS 1 respectively.

Claim 2. *If i and j are both even (or odd) and $|i-j| \geq 4$, then $\pi^1_i \neq \tau^1_j$.*

Proof. Assume that i and j are even. By Claim 1, the possible values of π^1_i are in $\{i-1, i, i+1, i+2\}$ and the possible values of τ^1_j are in $\{j-1, j, j+1, j+2\}$. Clearly $|i-j| \geq 4$ implies that $\pi^1_i \neq \tau^1_j$. Similarly the claim holds for the case when i and j are odd. □

The following claim shows that if the values of the i-th position of π and τ are different after running PASS 1, the difference will be kept (or the difference may be propagated to different position) after running the whole algorithm.

Claim 3. *If $\pi^1_i \neq \tau^1_i$ and $\pi_j = \pi^1_i$ for any i and j, then $\pi_j \neq \tau_j$.*

	x_{8k+i}	x_{8k+4+i}	x_{8k+8+i}	π_{8k+4+i}	π_{8k+8+i}
1	-	-	-	π^1_{8k+4+i}	π^1_{8k+8+i}
2	-	-	2	π^1_{8k+4+i}	$\pi^1_{8k+12+i}$
3	-	2	-	π^1_{8k+8+i}	π^1_{8k+4+i}
4	-	2	2	$\pi^1_{8k+12+i}$	π^1_{8k+4+i}
5	2	-	-	π^1_{8k+i}	π^1_{8k+8+i}
6	2	-	2	π^1_{8k+i}	$\pi^1_{8k+12+i}$
7	2	2	-	π^1_{8k+8+i}	π^1_{8k+i}
8	2	2	2	$\pi^1_{8k+12+i}$	π^1_{8k+i}

Table 3.2: Possible values of π_j after PASS 2 for $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{1, 2, 3, 4\}$.

Proof. Note that $\pi_j = \pi_i^1$ implies that $4|(j-i)$ since π_j must be one of the elements in $\{\pi^1_{i-8}, \pi^1_{i-4}, \pi^1_i, \pi^1_{i+4}, \pi^1_{i+8}\}$ by Claim 1. Similarly assume that $\tau_j = \tau^1_{i'}$, then we have $4|(j-i')$. Thus, $4|(i-i')$. If $|i-i'| \geq 4$, then we obtain $\pi^1_i \neq \tau^1_{i'}$ by Claim 2. Therefore, in this case, $\pi_j \neq \tau_j$. On the other hand, if $|i-i'| < 4$, it implies $i = i'$. By assumption, we have $\pi^1_i \neq \tau^1_i$ and this also implies $\pi_j \neq \tau_j$. □

Definition 1. *For any $i \neq j$, we say that position i can be covered with position j if $\delta(x_i, y_i) > \delta(\pi_i, \tau_i)$ and $\delta(x_j, y_j) < \delta(\pi_j, \tau_j)$, where $\delta(a, b) = 1$ if $a \neq b$ and 0 otherwise. (that is, $x_i \neq y_i$, $\pi_i = \tau_i$, $x_j = y_j$, and $\pi_j \neq \tau_j$). Furthermore, we say that position i is self-covered if $\delta(x_i, y_i) \leq \delta(\pi_i, \tau_i)$.*

For each i with $\delta(x_i, y_i) > \delta(\pi_i, \tau_i)$, it needs some other position to make up the decrease of distance at position i in order to satisfy the distance preserving property.

Definition 2. *Let* **NSC** *be the set of positions not self-covered, that is,* **NSC**$= \{i \in [n] : \delta(x_i, y_i) > \delta(\pi_i, \tau_i)\}$. *A covering pattern is a function* $g : [n] \to [n]$ *such that for any* $i \in$ **NSC**, $g(i)$ *covers* i *and for any* $i \in [n] \backslash$**NSC**, $g(i) = i$.

The following is our main claim which is crucial to show the distance-preserving property of algorithm A_{8n}.

Claim 4. *There exists a covering pattern* g *such that for any position* $j \in$**NSC**, $g(j) \in \{j-1, j-4, j-5, j-8, j-9\}$. *Furthermore,* $|g^{-1}(k) \cap \{k+1, k+4, k+5, k+8, k+9\}| \leq 1$ *for any position* k.

Proof. For any x and $y \in \{0,1\}^{8n}$, we define such a covering pattern g by analyzing every possible position $j \in [n]$ and setting $g(j)$ case by case.

Case 1 : [j with $x_j = y_j$] It implies that $\delta(x_j, y_j) = 0$, and it is always true that $\delta(\pi_j, \tau_j) \geq \delta(x_j, y_j)$. So j is self-covered. In this case,we can set $g(j) = j$.

Case 2 : [j with $x_j \neq y_j$ and one of x_j and y_j is 2] W.L.O.G., we may assume that $x_j = 2$ and $y_j \neq 2$.

- Case 2-1: [$j = 8k+4+i$ for some $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{1, 2, 3, 4\}$] Observe that in Table 3.2, under the case condition, the possible values of π_j are in $\{\pi^1_{8k+8+i}, \pi^1_{8k+12+i}\}$ and the possible values of τ_j are in $\{\tau^1_{8k+i}, \tau^1_{8k+4+i}\}$. Note that $\{\pi^1_{8k+8+i}, \pi^1_{8k+12+i}\} \cap \{\tau^1_{8k+i}, \tau^1_{8k+4+i}\} = \emptyset$ by Claim 2. Thus, $\pi_j \neq \tau_j$. So j is self-covered. In this case, we set $g(j) = j$.

- Case2-2: [$j = 8k+8+i$ for some $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{1, 2, 3, 4\}$] In this case, the possible values of π_j are in $\{\pi^1_{8k+i}, \pi^1_{8k+4+i}, \pi^1_{8k+12+i}\}$

and the possible values of τ_j are in $\{\tau^1_{8k+i}, \tau^1_{8k+4+i}, \tau^1_{8k+8+i}\}$. Assume that $\pi_j = \pi^1_{j_1}$ and $\tau_j = \tau^1_{j_2}$. If $j_1 \neq j_2$, then $|j_1 - j_2| \geq 4$ and $\pi_j \neq \tau_j$ by Claim 2. I.e., j is self-covered. In this case, set $g(j) = j$. On the other hand, if $j_1 = j_2$, then it must be the cases in row 3 and 4 (i.e. $j_1 = j_2 = 8k + 4 + i$) or in rows 7 and 8 (i.e. $j_1 = j_2 = 8k + i$) of Table 3.2. In both cases, observe that x_{8k+4+i} and y_{8k+4+i} must be 2 and $\pi_{8k+4+i} = \pi^1_{8k+12+i}$ and $\tau_{8k+4+i} = \tau^1_{8k+8+i}$. By Claim 2, $\pi_{8k+4+i} \neq \tau_{8k+4+i}$. Note that it's still possible $\pi_j \neq \tau_j$; i.e. j is self-covered, and we can simply set $g(j) = j$. So if $\pi_j \neq \tau_j$, then set $g(j) = j$, else $j = 8k + 8 + i$ can be covered with position $j - 4 = 8k + 4 + i$ and we set $g(j) = j - 4$.

For convenience, we can let $g(j) = j$ for all j by default. If j is not self-covered, then we can set $g(j)$ to be other value. In other words, we reset $g(j)$ whenever necessary.

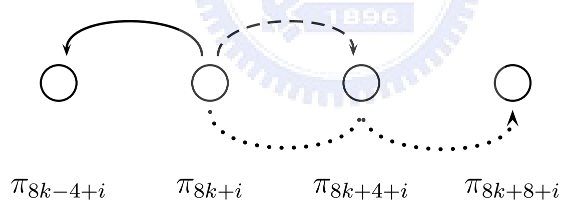

Figure 3.4: Possible final positions of π^1_{8k+i}, $i \in \{0, 1, 2, 3\}$.

Case 3 : [j with $x_j \neq y_j$ and $x_j, y_j \in \{0, 1\}$] In this case, W.L.O.G. we may assume $x_j = 1$ and $y_j = 0$. For convenience, we use Table 3.3 to show that the possible positions of π^1_{8k+i} and π^1_{8k+4+i}. For example, row 7 means that when $x_{8k-4+i} = 2$, $x_{8k+i} = 2$ and $x_{8k+4+i} \neq 2$, then after running PASS

	x_{8k-4+i}	x_{8k+i}	x_{8k+4+i}	π_{8k-4+i}	π_{8k+i}	π_{8k+4+i}	π_{8k+8+i}
1	-	-	-		π^1_{8k+i}	π^1_{8k+4+i}	
2	-	-	2		π^1_{8k+i}		π^1_{8k+4+i}
3	-	2	-		π^1_{8k+4+i}	π^1_{8k+i}	
4	-	2	2		π^1_{8k+4+i}		π^1_{8k+i}
5	2	-	-	π^1_{8k+i}		π^1_{8k+4+i}	
6	2	-	2	π^1_{8k+i}			π^1_{8k+4+i}
7	2	2	-	π^1_{8k+4+i}		π^1_{8k+i}	
8	2	2	2	π^1_{8k+4+i}			π^1_{8k+i}

Table 3.3: Possible final positions of π^1_{8k+i} and π^1_{8k+4+i}

2, π^1_{8k+i} will appear in position $8k+4+i$ (figure 3.4: dashed line) and π^1_{8k+4+i} in position $8k-4+i$.

- Case 3-1: [$\pi^1_j \neq \tau^1_j$ and $j = 8k+i$ for some $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{1, 2, 3, 4\}$] Note that $x_j \neq 2$. By Table 3.3, π^1_{8k+i} can be in position either $8k+i$ or $8k-4+i$. If $\pi_{8k+i} = \pi^1_{8k+i}$, then $\pi_{8k+i} \neq \tau_{8k+i}$ by Claim 3. Thus, j is self-covered and we set $g(j) = j$. Similarly it applies to the case when $\tau_{8k+i} = \tau^1_{8k+i}$. The rest of this case is that both π^1_{8k+i} and τ^1_{8k+i} are in position $8k-4+i$. When this happens, it implies that $x_{j-4} = y_{j-4} = 2$ by observing Table 3.3 and we have $\pi_{8k-4+i} = \pi^1_{8k+i}$ and $\tau_{8k-4+i} = \tau^1_{8k+i}$. By assumption that $\pi^1_{8k+i} \neq \tau^1_{8k+i}$, we conclude that j can be covered with $j-4$. In this case, set $g(j) = j-4$, if $\pi_j = \tau_j$.

- Case 3-2: [$\pi^1_j \neq \tau^1_j$ and $j = 8k+4+i$ for some $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{1, 2, 3, 4\}$] Again by observing Table 3.3 if $x_j \neq 2$ and $y_j \neq 2$, then

π^1_{8k+4+i} and τ^1_{8k+4+i} have three possible final positions i.e., $8k+4+i$, $8k+i$, and $8k-4+i$. We divide the analysis into three subcases.

– Subcase 3-2-I: [π^1_{8k+4+i} or τ^1_{8k+4+i} are in position $8k+4+i$] W.L.O.G. we assume that π^1_{8k+4+i} appears in position $8k+4+i$, i.e. $\pi_{8k+4+i} = \pi^1_{8k+4+i}$. By the assumption that $\pi^1_{8k+4+i} \neq \tau^1_{8k+4+i}$, we obtain $\pi_{8k+4+i} \neq \tau_{8k+4+i}$ by Claim 3. Thus $j = 8k+4+i$ is self-covered and set $g(j) = j$ by default.

– Subcase 3-2-II: [π^1_{8k+4+i} or τ^1_{8k+4+i} are in position $8k+i$] W.L.O.G. we assume that π^1_{8k+4+i} appears in position $8k+i$, i.e. $\pi_{8k+i} = \pi^1_{8k+4+i}$. We can assume that $\tau_{8k+4+i} \neq \tau^1_{8k+4+i}$, otherwise it has been done in Subcase 3-2-I. By Claim 3, it's clear that $\pi_{8k+i} \neq \tau_{8k+i}$. In this subcase since $\pi_{8k+4+i} \neq \pi^1_{8k+4+i}$, $\tau_{8k+4+i} \neq \tau^1_{8k+4+i}$ and both x_{8k+4+i} and y_{8k+4+i} are not equal to 2, it must be the cases in row 3 or row 7 of Table 3.3. In both cases we have $x_{8k+i} = y_{8k+i} = 2$. Thus j can be covered with $j-4$ and we set $g(j) = j-4$, if $\pi_j = \tau_j$.

– Subcase 3-2-III: [Both π^1_{8k+4+i} and τ^1_{8k+4+i} are in position $8k-4+i$] I.e. $\pi_{8k-4+i} = \pi^1_{8k+4+i}$ and $\tau_{8k-4+i} = \tau^1_{8k+4+i}$. Clearly $\pi_{8k-4+i} \neq \tau_{8k-4+i}$ by the assumption of Case 3-2 that $\pi^1_{8k+4+i} \neq \tau^1_{8k+4+i}$. Again, by observing Table 3.3, it must be the case that $x_{8k-4+i} = y_{8k-4+i} = 2$ and $x_{8k+i} = y_{8k+i} = 2$. Thus j can be covered with $j-8$ and we set $g(j) = j-8$, if $\pi_j = \tau_j$.

Next, we deal with the case that $\pi^1_j = \tau^1_j$ and $x_j, y_j \in \{0,1\}$ with $x_j \neq y_j$. By observing Table 3.1, in this case, j must be odd, and in rows 3 and 4 (i.e. $\pi^1_{2k+3} = \tau^1_{2k+3} = 2k+2$) or in rows 7 and 8 (i.e. $\pi^1_{2k+3} = \tau^1_{2k+3} = 2k+1$) in Table 3.1. Observe that $x_{j-1} = y_{j-1} = 1$

and $\pi_{j-1}^1 \neq \tau_{j-1}^1$ in these cases. We divide the analysis into two cases.

- Case 3-3: [$\pi_j^1 = \tau_j^1$ and $j = 8k + i$ for some $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{3, 5\}$] Note that $x_j \neq y_j$. From the above discussion, we know that $x_{j-1} = y_{j-1} = 1$ and $\pi_{j-1}^1 \neq \tau_{j-1}^1$. The possible final positions of π_{j-1}^1 and τ_{j-1}^1 are $j-1$ and $j-5$ by observing Table 3.3. Thus, there are the following three cases: (1) $\pi_{j-5} = \pi_{j-1}^1$ and $\tau_{j-1} = \tau_{j-1}^1$ (or symmetrically $\pi_{j-1} = \pi_{j-1}^1$ and $\tau_{j-5} = \tau_{j-1}^1$); (2)$\pi_{j-1} = \pi_{j-1}^1$ and $\tau_{j-1} = \tau_{j-1}^1$ and (3) $\pi_{j-5} = \pi_{j-1}^1$ and $\tau_{j-5} = \tau_{j-1}^1$. For (1), by Claim 3, $\pi_{j-1} \neq \tau_{j-1}$. Thus, j can be covered with position $j-1$ and we set $g(j) = j-1$. For (2), it is obvious that j can be covered with position $j-1$ and we set $g(j) = j-1$. For (3), note that $x_{j-5} = y_{j-5} = 2$ by observing Table 3.3. Thus j can be covered with position $j-5$ and we set $g(j) = j-5$.

- Case 3-4: [$\pi_j^1 = \tau_j^1$ and $j = 8k + 4 + i$ for some $k \in \{0, 1, \cdots, n-1\}$ and $i \in \{3, 5\}$] Again we have $x_{j-1} = y_{j-1} = 1$ and $\pi_{j-1}^1 \neq \tau_{j-1}^1$. By observing Table 3.3, the possible final positions of π_{j-1}^1 and τ_{j-1}^1 are $j-1$, $j-5$, and $j-9$. If one of the final positions of π_{j-1}^1 and τ_{j-1}^1 is $j-1$, then j can be covered with position $j-1$ by Claim 3 and we can set $g(j) = j-1$. Suppose that one of final positions is $j-5$. With the same argument as in Subcase 3-2-II, j can be covered with position $j-5$ and we can set $g(j) = j-5$. Finally, suppose that both the final positions are $j-9$. With the same argument as of Subcase 3-2-III, j can be covered with position $j-9$ and we can set $g(j) = j-9$.

By the above analysis, we can set up a covering pattern g such that $g(j) = j$ if position j is self-covered and $g(j) \in \{j-1, j-4, j-5, j-8, j-9\}$ for each $j \in \mathbf{NSC}$. Furthermore, we show that $|g^{-1}(k) \cap \{k+1, k+4, k+$

$5, k+8, k+9\}| \le 1$ for any position k. We illustrate this in Table 3.4.

covered case	necessary condition
$g(k+4) = k$	$x_k = y_k = 2$ $x_{k+4} \neq y_{k+4}$
$g(k+8) = k$	$x_k = y_k = 2$ $x_{k+4} = y_{k+4} = 2$ $x_{k+8} \neq y_{k+8}$
$g(k+1) = k$	$x_k = y_k = 1$ $x_{k+1} \neq y_{k+1}$
$g(k+5) = k$	$x_k = y_k = 2$ $x_{k+4} = y_{k+4} = 1$ $x_{k+5} \neq y_{k+5}$
$g(k+9) = k$	$x_k = y_k = 2$ $x_{k+4} = y_{k+4} = 2$ $x_{k+8} = y_{k+8} = 1$ $x_{k+9} \neq y_{k+9}$

Table 3.4: Necessary Conditions for Position Covering

In Table 3.4, we list the necessary conditions for the covering pattern g. Note that those conditions are all disjoint. This implies that $g^{-1}(k)$ contains at most one position in $\{k+1, k+4, k+5, k+8, k+9\}$ Therefore we complete the proof of Claim 4. □

Recall that **NSC**$= \{i \in [n] : \delta(x_i, y_i) > \delta(\pi_i, \tau_i)\}$. Based on Claim 4, we show that g on **NSC** is a one-to-one function.

Claim 5. *Let g be the covering pattern obtained in Claim 4. Then g :*

$\mathbf{NSC} \to [n]$ *is a one-to-one function and* $g(\mathbf{NSC}) \cap \mathbf{NSC} = \emptyset$*, and hence* $|g(\mathbf{NSC})| = |\mathbf{NSC}|$.

Proof. Assume that $g(i) = g(h) = j$. Thus we have $j \in \{i-1, i-4, i-5, i-8, i-9\} \cap \{h-1, h-4, h-5, h-8, h-9\}$. If $i \neq h$, then $|g^{-1}(j) \cap \{j+1, j+4, j+5, j+8, j+9\}| \geq 2$ since i and h are both in the intersection. However, this is impossible by Claim 4. Thus, $i = h$ and hence g is one-to-one. By Table 3.4, if k covers some other position, then $x_k = y_k$. By definition, if k can be covered with some other position, then $x_k \neq y_k$. Thus it implies $g(\mathbf{NSC}) \cap \mathbf{NSC} = \emptyset$. Since g is one-to-one, we have $|g(\mathbf{NSC})| = |\mathbf{NSC}|$. □

Now we show the distance-preserving property of A_{8n}. Note that for any $i \in \mathbf{NSC}$, $\delta(x_i, y_i) = 1$ and $\delta(\pi_i, \tau_i) = 0$. Also for any $i \in g(\mathbf{NSC})$, we have $\delta(x_i, y_i) = 0$ and $\delta(\pi_i, \tau_i) = 1$. Thus $\sum_{i \in \mathbf{NSC}} \delta(x_i, y_i) + \sum_{i \in g(\mathbf{NSC})} \delta(x_i, y_i) = \sum_{i \in \mathbf{NSC}} \delta(\pi_i, \tau_i) + \sum_{i \in g(\mathbf{NSC})} \delta(\pi_i, \tau_i)$ by Claim 5. Thus, we have

$$\begin{aligned} d_H(x, y) &= \sum_{i=1}^{8n} \delta(x_i, y_i) \\ &= \sum_{i \in \mathbf{NSC} \cup g(\mathbf{NSC})} \delta(x_i, y_i) + \sum_{i \notin \mathbf{NSC} \cup g(\mathbf{NSC})} \delta(x_i, y_i) \\ &\leq \sum_{i \in \mathbf{NSC} \cup g(\mathbf{NSC})} \delta(\pi_i, \tau_i) + \sum_{i \notin \mathbf{NSC} \cup g(\mathbf{NSC})} \delta(\pi_i, \tau_i) \\ &= \sum_{i=1}^{8n} \delta(\pi_i, \tau_i) = d_H(\pi, \tau). \end{aligned}$$

This completes the proof of Theorem 2. □

3.1.2 3-DPM$_H$ for input length ≥ 16

In this section, we modify our algorithm A_{8n} such that new algorithm can be applied to any input length at least 16. To achieve this goal, we need to show another property of algorithm A_{8n}. As in the previous section, let $\pi = A_{8n}(x)$ and π^1 be the intermediate result after PASS 1.

Lemma 1. *For any $i \in \{1, 2, \cdots, 8n\}$, $\pi_i \neq i - 3$.*

Proof. By way of contradiction, suppose that there is i such that $\pi_i = i - 3$. Assume that $\pi_i = \pi_j^1 = i - 3$ for some j. j must satisfy $4|(i - j)$. By the structure of PASS 1, $(i - 3) - 2 \leq j \leq (i - 3) + 2$. Thus, it must be the case that $j = i - 4$, that is $\pi_i = \pi_{i-4}^1 = i - 3$. If $\pi_i = \pi_{i-4}^1$, then we have $x_{i-4} = 2$. However, if $\pi_{i-4}^1 = i - 3$, then we have $x_{i-4} = 1$ by observing Table 3.1. Hence, we get a contradiction. □

Now we show the 3-DPM$_H$ A_{8n+k} as in Figure 3.5.

Algorithm A_{8n+k} $(8n \geq 16$ **,** $1 \leq k \leq 7)$ **:**
Input: $(x_1, \cdots, x_{8n+k}) \in Z_3^{8n+k}$
Output: $(\pi_1, \cdots, \pi_{8n+k}) \in S_{8n+k}$
$(\pi_1, \cdots, \pi_{8n}) \leftarrow A_{8n}(x_1, x_2 \cdots, x_{8n})$;
$(\pi_{8n+1}, \cdots, \pi_{8n+k}) \leftarrow (8n + 1, \cdots, 8n + k)$;
for $i = 1$ **to** k **do;**
  if $x_{8n+i} = 1$ **then swap** $(\pi_{8n+i}, \pi_{\pi^{-1}(i-3)})$;
  if $x_{8n+i} = 2$ **then swap** (π_{8n+i}, π_i);

Figure 3.5: 3-DPM$_H$ Algorithm A_{8n+k} for $k \in [7]$

We prove its correctness in the following theorem.

Theorem 3. *$A_{8n+k} : Z_3^{8n+k} \rightarrow S_{8n+k}$ is a* **3-DPM$_H$** *for all $n \geq 2$ and $k \in \{1, \cdots, 7\}$.*

Proof. Given two inputs $(x, w), (y, z) \in Z_3^{8n} \times Z_3^k$, suppose that $\pi = A_{8n+k}(x, w)$ and $\tau = A_{8n+k}(y, z)$. Let w^i and z^i denote the first i symbols of w and z respectively. Let π^i and τ^i be the permutations in S_{8n+i} obtained by running the i-th iteration in the for loop when the inputs are (x, w) and (y, z) respectively. It suffices to prove the following claim.

Claim 6. *$d_H((x, w^i), (y, z^i)) \leq d_H(\pi^i, \tau^i)$ for any $i \in \{0, \cdots, k\}$.*

Proof. We prove this claim by induction on i. It holds trivially for $i = 0$ since we have $d_H(x, y) \leq d_H(A_{8n}(x), A_{8n}(y)) = d_H(\pi^0, \tau^0)$. For the inductive step, suppose that $d_H(x, y) + d_H(w^{i-1}, z^{i-1}) \leq d_H(\pi^{i-1}, \tau^{i-1})$. We divide the analysis into the following cases.

- Case $[w_i = z_i]$: The claim holds trivially in this case since both swap operations in the ith iteration are the same.

- Case $[w_i \neq z_i$ and one of them is $0]$: W.L.O.G. we assume that $w_i = 0$. In this case we have $\pi^i_{8n+i} = 8n + i$, $\pi^i_{[1..8n+i-1]} = \pi^{i-1}$ and τ^i_{8n+i} equals to either $i - 3$ or τ^{i-1}_i. Thus we have $\delta(\pi^i_{8n+i}, \tau^i_{8n+i}) = 1$. W.L.O.G., we assume that $\tau^i_{8n+i} = \tau^{i-1}_i$ and hence $\tau^i_i = 8n + i$. So $\delta(\pi^i_i, \tau^i_i) = 1$. Also note that $\tau^i_t = \tau^{i-1}_t$ for any $t \in [8n + i - 1] \backslash \{i\}$. So we have $d_H((x, w^i), (y, z^i)) \leq d_H(\pi^i, \tau^i)$.

- Case $[w_i \neq z_i$, and $w_i, z_i \in \{1, 2\}]$: W.L.O.G. we assume that $w_i = 1$ and $z_i = 2$. In this case, $\pi^i_{8n+i} = i - 3$ and $\tau^i_{8n+i} = \tau^{i-1}_i = \tau_i$. By Lemma 1, we know that $\pi^{-1}(i - 3) \neq i$ and $\tau_i \neq i - 3$. Now it is easy to check $d_H(\pi^i, \tau^i) = d_H(\pi^{i-1}, \tau^{i-1}) + 1$. Hence we also have $d_H((x, w^i), (y, z^i)) \leq d_H(\pi^i, \tau^i)$.

□

Thus Theorem 3 follows from Claim 6. □

From Theorem 2 and Theorem 3, we give the first explicit construction of 3-DPM$_H$.

Corollary 4. *There exists an explicit construction of 3-DPM$_H$ from Z_3^n to S_n for any $n \geq 16$.*

Note that the above construction can be applied to the case when $q \geq 3$. However, for different q, we need a different version of lemma 1 in order to obtain an explicit construction of q-DPM$_H$.

3.2 Construction of PAs with Hamming Distance

As shown in [6] and [4], we know that distance-increasing mappings are quite helpful for constructing permutation arrays. Similarly we can make use of 3-DPM$_H$ to construct permutation array with hamming distance. In this section we introduce the construction and the corresponding encoding and decoding algorithms.

Theorem 4. *For all $N \geq 16$ and $d \leq N$, suppose C is an (N, d) ternary code. Then there is an (N, d) permutation array P with hamming distance and the same cardinality as C. If C has an efficient encoding/decoding algorithm pair, then there is an efficient encoding/decoding algorithm pair for P. Furthermore, if the decoding algorithm of C can correct up to e errors, then the decoding algorithm of P can decode correctly when the corrupted codeword π' satisfying $d_H(\pi, \pi') \leq e/4 - 2$, for some codeword $\pi \in P$.*

Proof. First note that C may not be a linear code. It can be any code over Z_3^N. Let $N = 8n + k$, $n \geq 2$ and $0 \leq k \leq 7$. By Theorem 3 and $n \geq 2$, we have a distance-preserving mapping $A_{8n+k} : Z_3^N \to S_N$. It is easy to see that $A_{8n+k}(C)$ is a permutation array of length N with minimum distance d. Let P be $A_{8n+k}(C)$ and so $|P| = |A_{8n+k}(C)| = |C|$.

Next consider the encoding issue. If C has an efficient encoding algorithm $E : Msg \to Z_3^N$, where Msg is any arbitrary message space with size equal to $|C|$. In particular, Msg usually is $Z_2^{log|C|}$ or $Z_3^{log_3|C|}$ when using ternary code. Let $E_P = A_{8n+k} \circ E$, then $E_P : Msg \to S_N$ is an efficient encoding algorithm for P because E and A_{8n+k} are both efficient.

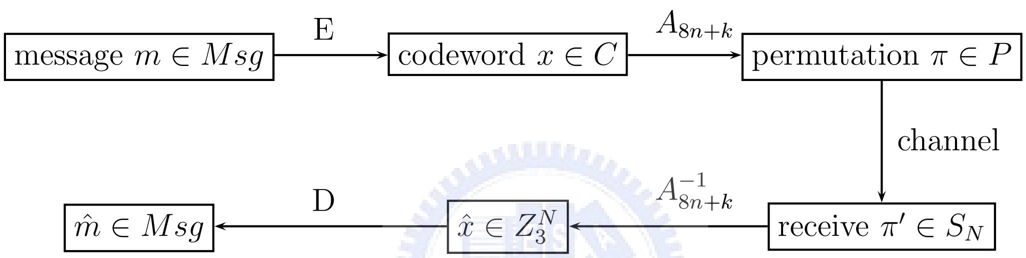

Figure 3.6: Construction of permutation array with 3-DPM$_H$

Finally consider the decoding issue. If C has an efficient decoding algorithm $D : Z_3^N \to C$ to correct up to e errors, i.e. for any codeword $x \in C$, and a corrupted codeword $y \in Z_3^N$ with $d_H(x, y) \leq e$, then $D(y) = x$. Let $\pi = A_{8n+k}(x) \in P$ and π' be a corrupted permutation satisfying $d_H(\pi, \pi') = d$. Without decoding π' to π directly, we design an algorithm A_{8n+k}^{-1} which compute the inversion of A_{8n+k}. If we can bound $d_H(A_{8n+k}^{-1}(\pi'), x)$ by $d_H(\pi, \pi')$, then we can decode P by combining A_{8n+k}^{-1} and D. We will describe how to do that in the rest of this proof.

To understand the decoding algorithm, let's give the idea first. Note that

A_{8n+k} is based on A_{8n} and then handle the last k positions. We consider the inversion of A_{8n} first. The idea is based on the proof of lemma 1. In the lemma, we prove for any i, $\pi_i \neq i-3$ by checking the path of symbol $i-3$ and derive a contradiction on the value of x_{i-4}. In general if the value of $\pi_i = t$ is given, we can determine the path of symbol t and the values of four positions of x. For example, given $\pi_5 = 12$, the path of symbol 12 can be determined as in the figure below, where symbol 12 goes to position 13 in PASS 1 and goes to position 9 and then 5 in PASS 2. Furthermore, we can determine that $x_{11} \neq 1$, $x_{12} = 1$, $x_5 = 2$ and $x_9 = 2$ by Tables 3.5 and 3.6.

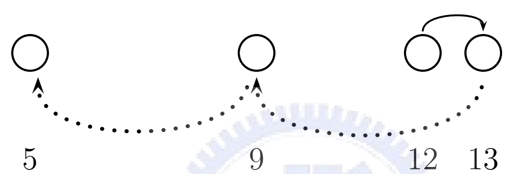

Figure 3.7: The unique path of symbol 12, given $\pi_5 = 12$.

We give Tables 3.5 and 3.6. for each possible value of π_i. For example, given $\pi_i = i+7$, $i \bmod 8 \in \{5,6,7,8\}$ and i is odd(the case $\pi_5 = 12$). In the gray area in Table 3.6, it implies that $\pi_i = \pi^1_{i+8}$, $\pi^1_{i+8} = i+7$ and determines the values of four positions of x, i.e., $x_i = 2$, $x_{i+4} = 2$, $x_{i+6} \neq 1$ and $x_{i+7} = 1$. One can verify each entry in the both tables by checking algorithm A_{8n}. Note that the tables give some entries, whcih are not applicable(n.a.) since under our construction certain positions in a permutation will avoid some values.

Checking each position of π, we can determine all the values of x_i, so we can compute the inversion of A_{8n}. Then we consider A^{-1}_{8n+k}. If given $\pi_i = t$ where $i \in [8n+1, 8n+k]$, then we set $x_i = 0$ if $\pi_i = i$, $x_i = 1$ if $\pi_i = i-3$, and

$i = 8k+8+j,\ j \in \{1,2,3,4\},\ i \in [n],\ k \in \{0, \frac{n}{8}-1\}$				
	i is odd		i is even	
	π_i		π_i	
π^1_{i-8} $x_{i-8}=2,$ $x_{i-4}=2$	$i-10$	$x_{i-10}=1,\ x_{i-9}=1$	$i-9$	$x_{i-9}=1,\ x_{i-8}\neq 1$
	$i-9$	$x_{i-10}\neq 1,\ x_{i-9}=1$	$i-8$	$x_{i-9}\neq 1,\ x_{i-8}\neq 1$
	$i-8$	$x_{i-9}\neq 1,\ x_{i-8}\neq 1$	$i-7$(n.a)	
	$i-7$	$x_{i-9}\neq 1,\ x_{i-8}=1$	$i-6$(n.a.)	
π^1_{i-4} $x_{i-8}\neq 2,$ $x_{i-4}=2$	$i-6$	$x_{i-6}=1,\ x_{i-5}=1$	$i-5$	$x_{i-5}=1,\ x_{i-4}\neq 1$
	$i-5$	$x_{i-6}\neq 1,\ x_{i-5}=1$	$i-4$	$x_{i-5}\neq 1,\ x_{i-4}\neq 1$
	$i-4$	$x_{i-5}\neq 1,\ x_{i-4}\neq 1$	$i-3$(n.a.)	
	$i-3$(n.a.)		$i-2$(n.a.)	
π^1_{i} $x_{i-4}\neq 2,$ $x_i\neq 2$	$i-2$	$x_{i-2}=1,\ x_{i-1}=1$	$i-1$	$x_{i-1}=1,\ x_i\neq 1$
	$i-1$	$x_{i-2}\neq 1,\ x_{i-1}=1$	i	$x_{i-1}\neq 1,\ x_i\neq 1$
	i	$x_{i-1}\neq 1,\ x_i\neq 1$	$i+1$	$x_i=1,\ x_{i+1}\neq 1$
	$i+1$	$x_{i-1}\neq 1,\ x_i=1$	$i+2$	$x_i=1,\ x_{i+1}=1$
π^1_{i+4} $x_{i-4}\neq 2,$ $x_i=2$	$i+2$	$x_{i+2}=1,\ x_{i+3}=1$	$i+3$	$x_{i+3}=1,\ x_{i+4}\neq 1$
	$i+3$	$x_{i+2}\neq 1,\ x_{i+3}=1$	$i+4$	$x_{i+3}\neq 1,\ x_{i+4}\neq 1$
	$i+4$	$x_{i+3}\neq 1,\ x_{i+4}\neq 1$	$i+5$	$x_{i+4}=1,\ x_{i+5}\neq 1$
	$i+5$	$x_{i+3}\neq 1,\ x_{i+4}=1$	$i+6$	$x_{i+4}=1,\ x_{i+5}=1$

Table 3.5: Path Table of π_i , $i = 8k+8+j,\ j \in \{1,2,3,4\}$

$i = 8k + 4 + j,\ j \in \{1, 2, 3, 4\},\ i \in [n],\ k \in \{0, \frac{n}{8} - 1\}$				
	i is odd		i is even	
	π_i		π_i	
π^1_{i-4} $x_i \neq 2$, $x_{i-4} = 2$	$i-6$	$x_{i-6} = 1,\ x_{i-5} = 1$	$i-5$	$x_{i-5} = 1,\ x_{i-4} \neq 1$
	$i-5$	$x_{i-6} \neq 1,\ x_{i-5} = 1$	$i-4$	$x_{i-5} \neq 1,\ x_{i-4} \neq 1$
	$i-4$	$x_{i-5} \neq 1,\ x_{i-4} \neq 1$	$i-3$(n.a.)	
	$i-3$(n.a.)		$i-2$(n.a.)	
π^1_{i} $x_i \neq 2$, $x_{i-4} \neq 2$	$i-2$	$x_{i-2} = 1,\ x_{i-1} = 1$	$i-1$	$x_{i-1} = 1,\ x_i \neq 1$
	$i-1$	$x_{i-2} \neq 1,\ x_{i-1} = 1$	i	$x_{i-1} \neq 1,\ x_i \neq 1$
	i	$x_{i-1} \neq 1,\ x_i \neq 1$	$i+1$	$x_i = 1,\ x_{i+1} \neq 1$
	$i+1$	$x_{i-1} \neq 1,\ x_i = 1$	$i+2$	$x_i = 1,\ x_{i+1} = 1$
π^1_{i+4} $x_i = 2$, $x_{i+4} \neq 2$	$i+2$	$x_{i+2} = 1,\ x_{i+3} = 1$	$i+3$	$x_{i+3} = 1,\ x_{i+4} \neq 1$
	$i+3$	$x_{i+2} \neq 1,\ x_{i+3} = 1$	$i+4$	$x_{i+3} \neq 1,\ x_{i+4} \neq 1$
	$i+4$	$x_{i+3} \neq 1,\ x_{i+4} \neq 1$	$i+5$	$x_{i+4} = 1,\ x_{i+5} \neq 1$
	$i+5$	$x_{i+3} \neq 1,\ x_{i+4} = 1$	$i+6$	$x_{i+4} = 1,\ x_{i+5} = 1$
π^1_{i+8} $x_i = 2$, $x_{i+4} = 2$	$i+6$	$x_{i+6} = 1,\ x_{i+7} = 1$	$i+7$	$x_{i+7} = 1,\ x_{i+8} \neq 1$
	$i+7$	$x_{i+6} \neq 1,\ x_{i+7} = 1$	$i+8$	$x_{i+7} \neq 1,\ x_{i+8} \neq 1$
	$i+8$	$x_{i+7} \neq 1,\ x_{i+8} \neq 1$	$i+9$	$x_{i+8} = 1,\ x_{i+9} \neq 1$
	$i+9$	$x_{i+7} \neq 1,\ x_{i+8} = 1$	$i+10$	$x_{i+8} = 1,\ x_{i+9} = 1$

Table 3.6: Path Table of π_i , $i = 8k + 4 + j,\ j \in \{1, 2, 3, 4\}$

$x_i = 2$ otherwise. If given $\pi_i = t$ where $i \in [1, 8n]$ and $t \in [8n+1, 8n+k]$, it implies π_i must have been swapped with $\pi_t = t$ in the final stage of algorithm A_{8n+k}. Thus we can just swap the value of π_i and π_t first and then determine x by the above approach.

We give the algorithm A^{-1}_{8n+k} as follows.

Algorithm A^{-1}_{8n+k} $(8n \geq 16\ ,\ k \in [0, 7])$:

(a) For all i in $[1, k]$, check whether π_{8n+i} is $8n + i$ or $i - 3$ or others, and then assign the corresponding value 0, 1, or 2, to x_{8n+i} respectively.

(b) For all i in $[1, 8n]$, if it is larger than $8n$, then swap (π_i, π_{π_i}).

(c) For each π_i, $i \in [1, 8n]$, let B_i is a bucket for index i. By the value of i and π_i, find the corresponding entries in Table 3.5 or Table 3.6. If it is not in the tables or not applicable(n.a), then do nothing. Else it will determine the values of four positions of x. Once we know $x_i = b \in \{1, 2\}$ by checking the tables, put b to B_i. If $x_i \neq b \in \{1, 2\}$, then put 0 to B_i.

(d) Decide x_i by a weighted majority vote. For each i in $[1, 8n]$, check B_i, '0' gives half weight, '1' and '2' each gives weight 1, and assign x_i be the largest weighted value $b \in \{0, 1, 2\}$. If tie, choose the larger value.

Let us explain the algorithm A^{-1}_{8n+k}. First using π_{8n+i} to decide x_{8n+i} for all $i \in [1, k]$. One can verify that if π_{8n+k} is not corrupted then x_{8n+k} is correct too. Next for $i \in [1, 8n]$, if $\pi_i > 8n$, it implies A_{8n+k} swap π_i and π_{π_i}, and then we should swap them. Third, the bucket B_i is designed to collect the vote (information) of x_i. For each $\pi_i = t$, one can determine the values of four positions of x by checking Table 3.5 and Table 3.6. And if it gives $x_i = b \in \{1, 2\}$ then puts b to B_i; if it gives $x_i \neq b \in \{1, 2\}$ then puts '0' to B_i. For example, if $\pi_5 = 12$, then it will put '2' to B_5 and B_9, put '1' to B_{12} and put '0' to B_{11}. Finally for each bucket B_i, make a weighted majority vote to decide the value of x_i. Because if it gives $x_i \neq 1$(or 2) then we puts 0 to bucket but the vote 0 does not guarantee x_i is 0, thus we give 0 half weight in the weighted majority vote. If tie, choose the larger value. Also we give another version of algorithm A_{8n+k}^{-1} in appendix A without table lookup.

Let's give figure 3.8 to illustrate the weighted majority vote. Each π' determine information in at most 4 positions of x. For each x_i, there are four positions of π' determine information of x_i if π' is not corrupted, since x_i can be used to decide whether to swap two positions or not in PASS 1, and to swap two positions or not in PASS 2. Thus it reveals some information about x_i by checking the path of those four symbols.

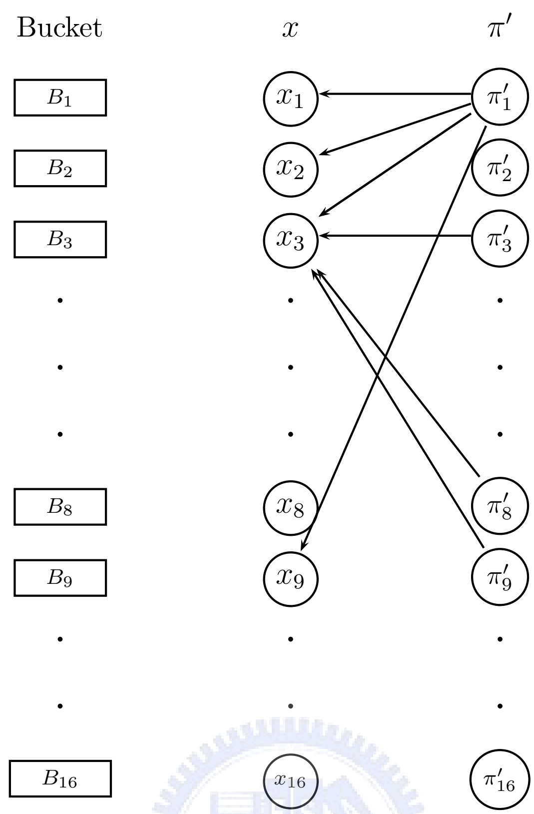

Figure 3.8: Weighted majority vote

The inverse algorithm A^{-1}_{8n+k} works well if π is not corrupted. Let us consider the corrupted π'. By Tables 3.5 and 3.6, each error will give us wrong information in at most 4 positions of x, and also lose correct information in at most 4 positions of x. It gives us a rough bound $d_H(A^{-1}_{8n+k}(\pi'), x) \leq 8 \cdot d_H(\pi, \pi')$. Here we give a better bound by analyzing it more carefully. Let $\pi = A_{8n+k}(x)$ be the correct codeword of x, and π' be the corrupted permutation.

Claim 7. $d_H(A^{-1}_{8n+k}(\pi'), x) \leq 4 \cdot d_H(\pi, \pi') + k$

Proof. Let $x' = A^{-1}_{8n+k}(\pi')$. First note $|B_i| = 4$ for all $i \in [1, 8n]$ if π' is not corrupted by figure 3.8. Furthermore it is easy to verify $B_i = \{2, 2, 0, 0\}$ if $x_i = 2$, $B_i = \{1, 1, 0, 0\}$ if $x_i = 1$ and $B_i = \{0, 0, 0, 0\}$ if $x_i = 0$. Observe that adding any extra vote to B_i or taking any vote from B_i would not change the result of the weighted majority vote. It implies that once the result of the weighted majority vote is wrong, then it must have at least two wrong votes. Each π_i votes at most four times and it creates at most eight changes on the buckets when π_i is corrupted since a wrong vote can have two effects, i.e., removing a vote from a bucket and adding an extra vote to another bucket.

Let's calculate the total influence of the corrupted positions.
Let $d_H(\pi_{[1,8n]}, \pi'_{[1,8n]}) = d_1$, $d_H(\pi_{[8n+1,8n+k]}, \pi'_{[8n+1,8n+k]}) = d_2$, and $d = d_1 + d_2$. It's clear $d_H(x_{[8n+1,8n+k]}, x'_{[8n+1,8n+k]}) \leq d_2$ because $\pi_i = \pi'_i$ implies $x_i = x'_i$ for $i \in [8n+1, 8n+k]$. For each $\pi_i \neq \pi'_i$ where $i \in [8n]$, by the above observation, it makes at most $8/2 = 4$ wrong decisions on the weighted majority vote. For each $i \in [8n]$ and $\pi_i = \pi'_i$, if $\pi'_i > 8n$, even π'_i is not corrupted, the corresponding $\pi'_{\pi'_i}$ could be corrupted already. Each corrupted $\pi'_{\pi'_i}$ adds wrong information to at most 8 buckets. But there are at most d_2 such $\pi'_{\pi'_i}$. Thus $d_H(x_{[1,8n]}, x'_{[1,8n]}) \leq 4 * (d_1 + d_2) = 4d$. And then $d_H(x, x') = d_H(x_{[1,8n]}, x'_{[1,8n]}) + d_H(x_{[8n+1,8n+k]}, x'_{[8n+1,8n+k]}) \leq 4d + k \leq 4 \cdot d_H(\pi, \pi') + k$.

□

Let us return to decoding issue. Let $D_P = D \circ A^{-1}_{8n+k}$ and $d \leq e/4 - 2$ be the number of errors in π'. By Claim 7, $d_H(x, x') \leq 4d + k \leq 4(e/4 - 2) + 7 \leq e$. Thus $D_P(\pi') = D(A^{-1}_{8n+k}(\pi')) = D(x') = x$ by the definition of D. We conclude that the decoding algorithm is efficient because D and A^{-1}_{8n+k} are both efficient and can correct up to $e/4 - 2$ errors. □

Note that we not only consider the corrupted codeword is a permutation $\pi' \in S_n$, but also consider the corrupted codeword is an n-ary vector $y \in \{1, 2, \cdots, n\}^n$. The decoding scheme can also decode correctly when the corrupted codeword y satisfying $d_H(\pi, y) \le e/4 - 2$, for some codeword $\pi \in P$.

In most cases, we take n as a multiple of 8 and then the decoding algorithm D_P guarantees to correct up to $e/4$ errors. In particular, the decoding algorithm can decode almost correctly as long as the errors does not exceed $e/4$ too much.

3.3 Previous Result and Comparison

Recall that $P_H(n, d)$ denotes the maximal size among all permutation arrays of length n and minimum distance d with hamming distance, and $A_q(n, d)$ the maximal size among all q-ary codes of length n and minimum distance d.

Corollary 5. *For all $n \ge 16$ and $d \le n$, $P_H(n, d) \ge A_3(n, d)$.*

Proof. Let C be the largest ternary code of length n with minimum distance d. By the definition of $A_3(n, d)$, $|C| = A_3(n, d)$. By Theorem 4, we have an (n, d) permutation array P with hamming distance of the same size as C. Thus $P_H(n, d) \ge |P| = |C| = A_3(n, d)$. □

Here we give some comparison between $A(n, d-k)$ and $A_3(n, d)$ for $k < d$. First of all, we need the well-known asymptotic Gilbert-Varshamov bound.

Fact 1. *(Theorem 2.10.8 in [13]) $A_3(n, d) \ge 3^{n(1-H_3(\frac{d}{n}))}$ for $d \le \frac{2n}{3}$ and sufficiently large n.*

The q-ary entropy function is defined as $H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x)$ for $0 < x \le 1$.

Thus for $d \leq \frac{3n}{5}$, we get a lower bound of $P_H(n,d) = 3^{\Omega(n)}$. On the other hand, $A(n, d-k) \leq 2^n$ for any k. Thus, in this case, we significantly improve the lower bounds by DIMs in [5].

Since the minimum input length of the known DIM_H which increases distance at least 2 is 16 (see [5]), we give a comparison between $A(16, d-2)$ and $A_3(n,d)$ in Table 3.7. Where the lower bound of $P_H(16,d)$ we obtained is much larger than the previous lower bound via DIM_H. Furthermore, we can decode the permutation arrays efficiently.

d	3	4	5	6	7	8	9	10	11	12	13	14
$L[A_3(16,d)]$	1062882	216513	19683	6561	729	297	253	54	18	9	4	3
$U[A(16,d-2)]$	65536	32768	3276	2048	340	256	37	32	6	4	2	2

Table 3.7: Comparison between $A_3(16,d)$ and $A(16,d-2)$ where $L[A_3(16,d)]$ stands for the lower bound of $A_3(16,d)$ as in [2] and $U[A(16,d-2)]$ the upper bound of $A(16,d-2)$ as in [1].

Chapter 4

Permutation Arrays with l_∞-Norm

In this chapter we give a construction of distance preserving mappings for binary vectors to permutations with l_∞-norm. It's similar to the previous chapter, first we give an algorithm to construct the DPM$_\infty$ and explain the recurrence construction. Next we describe how to construct PAs by the DPM$_\infty$ and introduce the encoding/decoding scheme of the PAs. Finally we give a direct way to construct permutation arrays. It's a general approach to constructing PAs for any n and d without using DPM$_\infty$.

4.1 DPMs from Z_2^{n-1} to S_n with l_∞-norm

Note that under l_∞-norm, the maximum distance between two permutations in S_n is at most $n-1$. It implies that DPM$_\infty$ from Z_2^n to S_n does not exist. So we consider DPM$_\infty$ from Z_2^{n-1} to S_n. We give an explicit algorithm to implement a family of DPM$_\infty$ in the following figure.

Theorem 5. *B_n is a DPM$_\infty$ from Z_2^{n-1} to S_n, for $n \geq 2$.*

Algorithm B_n **(** $n \geq 2$ **):**

Input: $(x_1, \cdots, x_{n-1}) \in Z_2^{n-1}$

Output: $(\pi_1, \cdots, \pi_n) \in S_n$

let $max = n; min = 1$;

for $i = 1$ **to** $n-1$ **do**

if $x_i = 1$

then $\pi_i = max$; $max = max - 1$;

else $\pi_i = min$; $min = min + 1$;

$\pi_n = min$;

Output $(\pi_1, \cdots, \pi_n)$.

Figure 4.1: Algorithm B_n computes DPM$_\infty$ from Z_2^{n-1} to S_n

Proof. Given $x \neq y \in Z_2^{n-1}$, let $\pi = B_n(x)$ and $\sigma = B_n(y)$. Assume k is the first position where x and y differ, i.e. $k = min\{j \in [1, n-1] : x_j \neq y_j\}$. W.L.O.G., let $x_k = 0$ and $y_k = 1$. Let $k_1 = |\{i \in [1, k-1] : x_i = 0\}|$ and $k_2 = |\{i \in [1, k-1] : y_i = 1\}|$. It's clear $k_1 + k_2 = k - 1$, $\pi_k = k_1 + 1$ and $\sigma_k = n - k_2$ by observing algorithm B_n. By the definition of k, $x_i = y_i$ for $i \in [1, k-1]$. Thus $d_H(x, y) \leq (n-1) - (k-1) = (n - k_2) - (k_1 + 1) = \sigma_k - \pi_k \leq l_\infty(\sigma, \pi)$. The first inequality holds because the length of x and y is $n-1$, and x and y have at least first $k-1$ bits equal. □

Observe that the algorithm gives a recursive construction for more general DPM$_\infty$ or DIM$_\infty$. We define the general (n_1, n_2, k)-DIMs and the corresponding algorithm C as follows.

Definition 3. *Given $n_1, n_2 \in N$, $k \in Z$ and d a distance metric, an (n_1, n_2, k)-DIM$_d$ is a mapping $f : Z_2^{n_1} \to S_{n_2}$ such that for any $x, y \in Z_2^{n_1}$, $x \neq y$, $d(f(x), f(y)) \geq d_H(x, y) + k$.*

Note that in the above definition n_1 and n_2 can be different, and k can be negative to indicate the distance decreasing mappings. In our definition, $(n, n, 1)$-DIM$_H$ is different from the (n,1)-DIM in [4]. (The latter relaxes the inequality $d(f(x), f(y)) \geq d_H(x, y) + k$ if $d(f(x), f(y))$ is already the maximum distance, i.e, $\max_{\pi,\sigma \in S_n} d(\pi, \sigma)$.)

Algorithm C**:**

Input: $(x_1, \cdots, x_{n_1+1}) \in Z_2^{n_1+1}$, f is an (n_1, n_2, k)-DIM$_\infty$

Output: $(\pi_1, \cdots, \pi_{n_2+1}) \in S_{n_2+1}$

if $x_1 = 0$

then $\pi_1 = 1$;

for $i = 2$ **to** $n_2 + 1$ **do** $\pi_i = f(x_{[2,n_2+1]})_{i-1} + 1$;

else $\pi_1 = n_2 + 1$;

for $i = 2$ **to** $n_2 + 1$ **do** $\pi_i = f(x_{[2,n_2+1]})_{i-1}$;

Output $(\pi_1, \cdots, \pi_{n_2+1})$.

Figure 4.2: Algorithm C computes an $(n_1 + 1, n_2 + 1, k)$-DIM$_\infty$ with an (n_1, n_2, k)-DIM$_\infty$

Theorem 6. *Given an (n_1, n_2, k)-DIM$_\infty$, then an $(n_1 + 1, n_2 + 1, k)$-DIM$_\infty$ can be obtained by algorithm C.*

Proof. Let f be an (n_1, n_2, k)-DIM$_\infty$. Suppose $x \neq y \in Z_2^{n_1+1}$ and let $\pi = C(x)$ and $\sigma = C(y)$. Note that $n_1 \leq n_2 - 1 - k$ because n_1 is the maximum distance among $Z_2^{n_1}$ and $n_2 - 1$ is the maximum distance among S_{n_2} and the existence of (n_1, n_2, k)-DIM$_\infty$. Thus if $x_1 = 0$ and $y_1 = 1$, then $d_H(x, y) \leq n_1 + 1 \leq n_2 - k = d_\infty(\sigma, \pi) - k$. The last equality holds because $\pi_1 = 1$ and $\sigma_1 = n_2 + 1$. It's similar for $x_1 = 1$ and $y_1 = 0$. For the case $x_1 = 0$ and

$y_1 = 0$, we have

$$\begin{aligned}
d_\infty(\sigma,\pi) &= \max_{i\in[2,n_2+1]}(\pi_i - \sigma_i) \qquad (\because \pi_1 = \sigma_1 = 1) \\
&= \max_{i\in[2,n_2+1]}((f(x_{[2,n_2+1]})_{i-1}+1) - (f(y_{[2,n_2+1]})_{i-1}+1)) \\
&= \max_{i\in[1,n_2]}(f(x_{[2,n_2+1]})_i - f(y_{[2,n_2+1]})_i) \\
&= d_\infty(f(x_{[2,n_2+1]}), f(y_{[2,n_2+1]}))
\end{aligned}$$

Thus $d_H(x,y) = d_H(x_{[2,n_2+1]}, y_{[2,n_2+1]}) \le d_\infty(x_{[2,n_2+1]}, y_{[2,n_2+1]}) - k = d_\infty(\pi,\sigma) - k$. It's similar for the case $x_1 = 1$ and $y_1 = 1$. □

Repeat the above, we have the following corollary.

Corollary 6. *Given an (n_1, n_2, k)-DIM$_\infty$, then (n_1+n, n_2+n, k)-DIM$_\infty$ can be constructed for all $n \in N$.*

In fact algorithm B_n is a special case of the recurrence construction with a basis case $(1,2,0)$-DIM$_\infty$ which maps $0 \in Z_2^1$ to $(12) \in S_2$ and $1 \in Z_2^1$ to $(21) \in S_2$. In general, to find a basis construction is a very time-consuming job, since the search space of mapping : $Z_2^{n_1} \to S_{n_2}$ is $\binom{n_2!}{2^{n_1}}$, which can be very large even for small n_1 and n_2. Once the basis construction is available, Algorithm C will systematically generate larger constructions. We show $(4,4,-1)$-DIM$_\infty$ in Table 1, which can be found by back-tracking search.

4.2 Encoding and Decoding with PAs by DPM$_\infty$

Similar to the hamming distance metric, we can construct permutation array with l_∞-norm by DPM$_\infty$. We give the decoding algorithms in figure 4.4.

Theorem 7. *Let C be an $(n-1,d)$ binary code. Then there is an (n,d) permutation array P with l_∞ norm such that $|P| = |C|$. If C has efficient encod-*

x	f(x)	x	f(x)	x	f(x)	x	f(x)
0000	1234	0001	1243	0010	1324	0011	1342
0100	1423	0101	1432	0110	2314	0111	2341
1000	2134	1001	2143	1010	3124	1011	3142
1100	3214	1101	3241	1110	2413	1111	2431

Table 4.1: f is a $(4, 4, -1)$-DIM$_\infty$.

ing/decoding algorithms, then P has efficient encoding/decoding algorithms. Furthermore, if the decoding algorithm of C can correct up to e errors, then the decoding algorithm of P can decode correctly when the corrupted codeword π' satisfying $d_\infty(\pi, \pi') \leq e/2$, for some codeword $\pi \in P$.

Proof. First note that C may not be a linear code. It can be any code over Z_2^{n-1}. By Theorem 5, we have a DPM$_\infty$ $B_n : Z_2^{n-1} \to S_n$. It is easy to see that $B_n(C)$ is a permutation array of length n with minimum distance d. Let P be $B_n(C)$ and so $|P| = |B_n(C)| = |C|$.

Next consider the encoding issue. Let $E : Msg \to Z_2^n$ be an efficient encoding algorithm for C, where Msg is any arbitrary message space with size equal to $|C|$. Typically, Msg can be $Z_2^{\lfloor log|C| \rfloor}$, but we consider the general case here. Let $E_P = B_n \circ E$, where B_n is a linear-time algorithm as in figure 4.1. Then $E_P : Msg \to S_n$ is an efficient encoding algorithm for P because E and B_n are both efficient.

Finally consider the decoding issue. If C has an efficient decoding algorithm $D : Z_2^{n-1} \to Msg$, which can correct up to e errors, i.e., for any message $m \in Msg$, and a corrupted codeword $y \in Z_2^{n-1}$ with $d_H(E(m), y) \leq e$, then $D(y) = m$. Let $x = E(m) \in C$, $\pi = B_n(x) \in P$ and $\pi' \in S_n$ be a corrupted permutation satisfying $d_\infty(\pi, \pi') \leq d$. The decoding idea is that first trans-

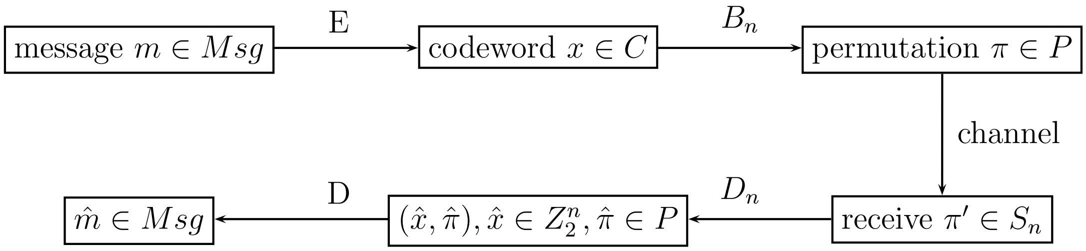

Figure 4.3: Encoding and decoding with DPM_∞

form π' to $\hat{x}$, which is an estimation of x, and then use D to recover m with $\hat{x}$.

Note that π have a good property that if $x[1, i-1]$ is fixed then π_i has only two possible values. We state it formally in the following claim.

Claim 8. *Let $x \in Z_2^{n-1}$ and $\pi = B_n(x)$. Assume there are t 0's in $x_{[1,i-1]}$. If $x_i = 0$ then $\pi_i = t + 1$ else $\pi_i = n - i + 1 + t$.*

Proof. Observe in algorithm B_n, $max = n$ and $min = 1$ initially. In round j, if $x_j = 0$ then the value of min increases by 1, otherwise max decreases by 1. After the $(i-1)$-st iteration, $min = 1 + t$ and $max = n - ((i-1) - t) = n - i + 1 + t$, where $(i-1) - t$ is the number of 1's in $x_{[1,i-1]}$. Thus in the i th iteration, If $x_i = 0$ then $\pi_i = min = t + 1$ else $\pi_i = max = n - i + 1 + t$. □

By claim 1, we can sequentially determine $\hat{x}_i$ from π'_i by a very simple rule. Initially π_1 has two possible values, i.e., 1 or n. We decide $\hat{x}_1$ by which value π'_1 is closer to. If π'_1 is closer to 1, we set $\hat{x_i} = 0$; otherwise we set $\hat{x_i} = 1$. Based on $\hat{x}_{[1,i-1]}$ and claim 8, there are two possible values for π'_i. Repeat the procedure we obtain $\hat{x}$. We give the decoding algorithm D_n in figure 4.4.

Claim 9. *For any $i \in [1, n-(2d+1)]$, $\hat{x}_i = x_i$ and $\hat{\pi}_i = \pi_i$*

Algorithm D_n:

Input: $(\pi'_1, \cdots, \pi'_n) \in S_n$

Output: $(\hat{x}_1, \cdots, \hat{x}_{n-1})$

let $max = n, min = 1$;

for $i = 1$ **to** $n-1$ **do**

if $|max - \pi'_i| < |\pi'_i - min|$

then $\{\hat{x}_i = 1;\ \hat{\pi}_i = max;\ max \leftarrow max - 1;\}$

else $\{\hat{x}_i = 0;\ \hat{\pi}_i = min;\ min \leftarrow min + 1;\}$

$\hat{\pi}_n = min$;

Output $\hat{x}$.

Figure 4.4: Algorithm D_n

Proof. We prove it by induction on i. The basis case has been explained above. Assume that $\hat{\pi}_{[1,i-1]} = \pi_{[1,i-1]}$ and $\hat{x}_{[1,i-1]} = x_{[1,i-1]}$ where $i \in [1, n-(2d+1)]$. By claim 8 and the induction hypothesis, assume t is the number of 0's in $x_{[1,i-1]}$. Then the only two possible values for π_i are $t+1$ or $n-i+1+t$. Let $min = t+1$ and $max = n-i+1+t$ be the value in the algorithm after the $(i-1)$-th iteration. If $\pi_i = min$, then $\pi'_i - min = \pi'_i - \pi_i \le d_\infty(\pi', \pi) \le d$ by the assumption. And $max - \pi'_i \ge (n-i+1+t) - (min+d) = n-i-d \ge n-(n-(2d+1))-d) = d+1$, where the last inequality holds by $i \le n-(2d+1)$. Thus the decoding is correct, since π'_i is closer to min than max. It implies $\hat{\pi}_i = \pi_i$ and $\hat{x}_i = x_i$. Similarly, it follows for $\pi_i = max = n-i+1+t$.

□

Finally we complete the decoding algorithm by combining D and D_n. First we get $\hat{x}$ by $D_n(\pi')$ and then output $D(\hat{x})$. Let $d_\infty(\pi, \pi') = d$ and $d \le e/2$. By claim 9, $d_H(x, \hat{x}) = d_H(x_{[n-2d,n-1]}, \hat{x}_{[n-2d,n-1]}) \le 2d \le e$, thus $\hat{x}$

can be decoded correctly to m by the definition of D. We conclude that the decoding algorithm is efficient because D and D_n are both efficient and can decode correctly while the corrupted codeword π' satisfying $d_\infty(\pi, \pi') \leq e/2$. □

Note that for any $(n-1, d)$ code, it can only decode correctly up to $(d-1)/2$ errors with uniquely decoding, and then the decoding algorithm of P is only proven to decode π' where $d_\infty(\pi, \pi') \leq (d-1)/4$ by the above construction.

It's similar to the last chapter. We can not only consider the corrupted codeword is a permutation $\pi' \in S_n$, but also consider the corrupted codeword is an n-ary vector $y \in \{1, 2, \cdots, n\}^n$. The decoding scheme can also decode correctly when the corrupted codeword y satisfying $d_H(\pi, y) \leq e/2$, for some codeword $\pi \in P$.

4.3 Encoding and Decoding Directly with PAs under l_∞-norm

Note that claim 9 implies the decoding errors only occur in the last $2d$ positions of $\hat{x}$. It means that we don't need a good code to do the mapping. Instead, one can directly encode and decode using the first $n-d$ position, where d is the minimal distance we want for the permutation array. We show the algorithm in figure 4.6.

The algorithm is very similar to B_n in the first loop and the second loop is not important– just fill legal values to $\pi_{[n-d+1,n]}$. The algorithm is an encoding algorithm for an (n, d) permutation array under l_∞-norm. Let's state it in the following theorem.

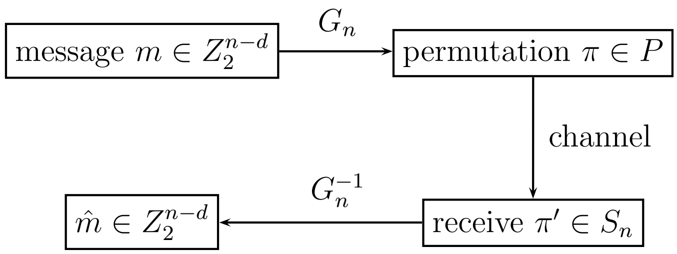

Figure 4.5: Direct encoding and decoding scheme with PAs

Algorithm G_n**:**

Input: $(x_1, \cdots, x_{n-d}) \in Z_2^{n-d}$

Output: $(\pi_1, \cdots, \pi_n) \in S_n$

  let $max = n, min = 1$;

  for $i = 1$ **to** $n - d$ **do**

    if $x_i = 1$

      then $\pi_i = max$;

          $max \leftarrow max - 1$;

      else $\pi_i = min$;

          $min \leftarrow min + 1$;

  for $i = n - d + 1$ **to** n **do**

    $\pi_i = min$;

    $min \leftarrow min + 1$;

Output $(\pi_1, \cdots, \pi_n)$.

Figure 4.6: Algorithm G_n encodes from Z_2^{n-d} to S_n

Theorem 8. *The range of $G_n : Z_2^{n-d} \to S_n$ is an (n,d) permutation array P with l_∞-norm.*

Proof. Given $x \neq y \in Z_2^{n-d}$, let $\pi = G_n(x)$ and $\sigma = G_n(y)$. Assume k is the first position where x and y differ, i.e. $k = min\{j \in [1, n-d] : x_j \neq y_j\}$. W.L.O.G., let $x_k = 0$ and $y_k = 1$.

Let $k_1 = |\{i \in [1, k-1] : x_i = 0\}|$ and $k_2 = |\{i \in [1, k-1] : y_i = 1\}|$. It's clear $\pi_k = k_1 + 1$ and $\sigma_k = n - k_2$ by observing algorithm G_n. By the definition of k, $x_i = y_i$ for $i \in [1, k-1]$, so $k_1 + k_2 = k - 1$. Thus $d_\infty(\sigma, \pi) \geq \sigma_k - \pi_k = (n - k_2) - (k_1 + 1) = n - k \geq n - (n-d) = d$. The last inequality holds because $k \leq n - d$, which is the length of x. Thus the range of G_n is a (n,d) permutation array under l_∞-norm, and G_n is a direct encoding algorithm for message space Z_2^{n-d}. □

The decoding algorithm G_n^{-1} is also very similar to algorithm D_n as in figure 4.4.

Theorem 9. *For any n and $d < n$, if $\pi = G_n(x)$ is the codeword, π' is a corrupted codeword such that $d_\infty(\pi, \pi') \leq (d-1)/2$ then $G_n^{-1}(\pi') = x$.*

Proof. We prove it by induction on i. Let x' be the output of $G_n^{-1}(\pi')$. The basis case is clear because $\pi_1 = 1$ or n. By the assumption $d_\infty(\pi, \pi') \leq \frac{d-1}{2}$, x_1 must be 0 if $\pi'_1 \leq 1 + \frac{d-1}{2}$; x_1 must be 1 if $\pi'_1 \geq n - \frac{d-1}{2}$. Assume that $x'_{[1,i-1]} = x_{[1,i-1]}$ where $i \in [2, n-d]$. By the same argument as for claim 8 and the induction hypothesis, assume t is the number of 0's in $x_{[1,i-1]}$, and then the only two possible values of π_i are $t+1$ or $n-i+1+t$. Let $min = t+1$ and $max = n-i+1+t$ after the $(i-1)$-th iteration. If $\pi_i = min$, $\pi'_i - min = \pi'_i - \pi_i \leq l_\infty(\pi', \pi) \leq \frac{d-1}{2}$ by the assumption. And $max - \pi'_i \geq (n-i+1+t) - (min + \frac{d-1}{2}) = n - i - \frac{d-1}{2} \geq n - (n-d) - \frac{d-1}{2} = \frac{d+1}{2}$, where the last inequality holds by $i \leq n-d$. Thus the decoding is correct,

Algorithm G_n^{-1}:

Input: $(\pi_1, \cdots, \pi_n) \in S_n$

Output: $(x_1, \cdots, x_{n-d})$

  let $max = n, min = 1$;

  for $i = 1$ **to** $n - d$ **do**

    if $|max - \pi_i| < |\pi_i - min|$

      then $\{x_i = 1; \hat{\pi}_i = max; max \leftarrow max - 1;\}$

      else $\{x_i = 0; \hat{\pi}_i = min; min \leftarrow min + 1;\}$

Output $(x_1, \cdots, x_{n-d})$

Figure 4.7: Algorithm G_n^{-1} is the decoding algorithm for G_n.

since π'_i is closer to min than max. It implies $x'_i = x_i$. The proof is similar for $\pi_i = max = n - i + 1 + t$.

□

Corollary 7. *There exists an (n, d) permutation array P under l_∞-norm, $|P| = 2^{n-d}$, and P have efficient encoding/decoding algorithms. Furthermore, the decoding algorithm of P can decode correctly while the corrupted codeword π' satisfying $d_\infty(\pi, \pi') \leq (d - 1)/2$.*

It's the same as the binary code, for any (n, d) permutation array P with metric d, it can be only uniquely decoded correctly as long as the corrupted distance $\leq (d - 1)/2$.

The decoding algorithm G_n^{-1} is proven to decode correctly while the corrupted distance $\leq (d - 1)/2$. But the construction of an (n, d) PA in the previous section requires an $(n - 1, d)$ binary code and just have a decoding algorithm proven to decode distance $\leq (d - 1)/4$. By the well known singleton bound in [13], $A(n-1,d) \leq 2^{n-d}$, and by Corollary 7.4.4 in [13], the equality does not hold for $d \neq 1$ or n. It implies that the direct construction in this section has a better codeword size and decoding algorithm than the construction via DPM_∞ in the last section.

(n,d) PA	codeword size	efficient encoder	efficient decoder
Gilbert Bound	$\frac{n!}{[(2d-1)!]^{\frac{n}{2d-1}}}$	May not exist	May not exist
DPM_∞ and $(n-1,d)$ code C	$\|C\| \leq A(n-1,d)$	$E_P = B_n \circ E$	$d(\pi',\pi) \leq (d-1)/4$
Construct directly	2^{n-d}	G_n	$d(\pi',\pi) \leq (d-1)/2$

Table 4.2: Comparison of the three constructions

In general, any construction via DPM/DIM has a disadvantage that the decoding algorithm would uses the decoding algorithm of the code C, where C is the code it applies. But it results that we need to decode twice and lose some decoding power.

Chapter 5

Conclusion and Open Problem

We first prove the permutation array version of Gilbert bound. It gives a lower bound for $P_f(n, d)$ for any metric f but those permutation arrays may not have efficient encoding/decoding algorithms. The main result of this thesis is constructing the permutation arrays with efficient encoding/decoding algorithms based on Hamming distance and l_∞-norm.

For the hamming distance, we give the first explicit construction of 3-DPM_H, and one can construct PAs under hamming distance, and these PAs have efficient encoding/decoding algorithms. It significantly improves the size of (n, d) permutation array asymptotically in [5] Moreover, following the same paradigm, one can obtain q-DPM_H for all $q > 3$. There is an open question that how to construct 3-DIM_H.

For the l_∞-norm, we give the first explicit construction of distance-preserving mappings from binary vectors to permutations with l_∞-norm. By DPM_∞, one can construct PAs under l_∞-norm, and these PAs have efficient encoding/decoding algorithms. We also propose a direct construction of (n, d) permutations arrays under l_∞-norm whose cardinality equals 2^{n-d}. These PAs have efficient encoding/decoding algorithms. Furthermore, the decoding algorithm can decode correctly if the distance between the corrupted codeword and the correct codeword is at most $(d-1)/2$.

It's interesting to consider other distance metrics, such as l_1-norm or l_2-norm. We leave it as an open question.

Appendix A

Algorithm A_{8n+k}^{-1} without table lookup

We give the algorithm A_{8n+k}^{-1} without table lookup in figureA.1.

Algorithm A^{-1}_{8n+k} without table lookup ($8n \geq 16$ **,** $k \in [0,7]$) **:**
Input: $(\pi_1, \cdots, \pi_{8n+k}) \in S_{8n+k}$
Output: $(x_1, \cdots, x_{8n+k}) \in Z_3^{8n+k}$
$(x_1, \cdots, x_{8n+k}) \leftarrow (0, 0, \cdots, 0)$**;**
$B_1, B_2, \cdots, B_{8n}$ **are** $8n$ **empty buckets ;**
for $i = 1$ **to** k **do;**
  if $\pi_{8n+i} = 8n + i$ **then** $x_{8n+i} \leftarrow 0$ **;**
  if $\pi_{8n+i} = i - 3$ **then** $x_{8n+i} \leftarrow 1$ **;**
  if $\pi_{8n+i} \neq 8n + i$ **and** $\pi_{8n+i} \neq i - 3$ **then** $x_{8n+i} \leftarrow 2$ **;**

for $i = 1$ **to** $8n$ **do;**
  if $\pi_i > 8n$ **then swap** (π_i, π_{π_i}) **;**
  let $t = \pi_i$**;**
  if $i\%2 = 1$ **then let** $p = \{t-1, t, t+1, t+2\} \cap \{i-8, i-4, i, i+4, i+8\}$ **;**
      else let $p = \{t-2, t-1, t, t+1\} \cap \{i-8, i-4, i, i+4, i+8\}$ **;**

  if $i\%2 = 1$
   then if $\pi_i = p + 1$ **then put** 0 **to** B_{p-1} **, put** 1 **to** B_p **;**
     if $\pi_i = p$ **then put** 0 **to** B_{p-1} **, put** 0 **to** B_p **;**
     if $\pi_i = p - 1$ **then put** 0 **to** B_{p-2} **, put** 1 **to** B_{p-1} **;**
     if $\pi_i = p - 2$ **then put** 1 **to** B_{p-2} **, put** 1 **to** B_{p-1} **;**

  if $i\%2 = 0$
   then if $\pi_i = p + 2$ **then put** 1 **to** B_p **, put** 1 **to** B_{p+1} **;**
     if $\pi_i = p + 1$ **then put** 1 **to** B_p **, put** 0 **to** B_{p+1} **;**
     if $\pi_i = p$ **then put** 0 **to** B_{p-1} **, put** 0 **to** B_p **;**
     if $\pi_i = p - 1$ **then put** 1 **to** B_{p-1} **, put** 0 **to** B_p **;**

  if $i\%8 \in [1,4]$
   then if $p = i + 4$ **then put** 0 **to** B_{i-4} **, put** 2 **to** B_i **;**
     if $p = i$ **then put** 0 **to** B_{p-4} **, put** 0 **to** B_i **;**
     if $p = i - 4$ **then put** 0 **to** B_{i-8} **, put** 2 **to** B_{i-4} **;**
     if $p = i - 8$ **then put** 2 **to** B_{i-8} **, put** 2 **to** B_{i-4} **;**

  if $i\%8 \in [5,8]$
   then if $p = i + 8$ **then put** 2 **to** B_i **, put** 2 **to** B_{i+4} **;**
     if $p = i + 4$ **then put** 2 **to** B_i **, put** 0 **to** B_{i+4} **;**
     if $p = i$ **then put** 0 **to** B_{i-4} **, put** 0 **to** B_i **;**
     if $p = i - 4$ **then put** 2 **to** B_{i-4} **, put** 0 **to** B_i **;**

 for $i = 1$ **to** $8n$ **do decide** x_i **by majority vote of** B_i**.**
  (0 **gives half weight, if tied choose the larger value)**
end

Figure A.1: Algorithm A^{-1}_{8n+k} without table lookup for $n \geq 2$, $k \in [0,7]$

By checking the tables 3.5, 3.6, one can find the intermediate position of t given $\pi_i = t$. We state it formally by the following claim.

Claim 10. *Let $\pi_i = t$ and let $p = \{t-1, t, t+1, t+2\} \cap \{i-8, i-4, i, i+4, i+8\}$ if i is odd, $p = \{t-2, t-1, t, t+1\} \cap \{i-8, i-4, i, i+4, i+8\}$ if i is even. Then $\pi^1_p = t$ and $\pi_i = \pi^1_p$.*

The claim is true clearly by checking all the entries of Table 3.5 and Table 3.6. For example, given $\pi_i = i+7$, $i \bmod 8 \in \{5,6,7,8\}$ and i is odd. Then $p = \{t-1, t, t+1, t+2\} \cap \{i-8, i-4, i, i+4, i+8\} = \{i+6, i+7, i+8, i+9\} \cap \{i-8, i-4, i, i+4, i+8\} = i+8$. It's the gray area in the second table.

Proof. Assume j is the index such that $\pi^1_j = t$ and $\pi_i = \pi^1_j$. Consider when i is odd first. By Claim1, $t \in \{j-2, j-1, j, j+1\}$. It implies $j \in \{t-1, t, t+1, t+2\}$. By Claim 1 again, $\pi_i \in \{\pi^1_{i-8}, \pi^1_{i-4}, \pi^1_i, \pi^1_{i+4}, \pi^1_{i+8}\}$. It implies $j \in \{i-8, i-4, i, i+4, i+8\}$ since $\pi_i = \pi^1_j$. Thus $j \in \{i-8, i-4, i, i+4, i+8\} \cap \{t-1, t, t+1, t+2\}$. It implies $j = p$ uniquely and so $\pi^1_p = t$ and $\pi_i = \pi^1_p$. The proof is similar for even t.

□

The claim gives us the algorithm A^{-1}_{8n+k} in figure A.1 without using table lookup.

Bibliography

[1] E. Agrell, A. Vardy, K. Zeger, A table of upper bounds for binary codes. *IEEE Transactions on Information Theory*, 47(7): 3004-3006, Nov. 2001.

[2] A.E. Brouwer, H.O. Hamalainen, P.R.J. Ostergard, N.J.A. Sloane, Bounds on mixed binary/ternary codes. *IEEE Transactions on Information Theory*, 44(1): 140-161, Jan. 1998.

[3] H. Chadwick, L. Kurz, Rank permutation group codes based on Kendall's correlation statistic. *IEEE Transactions onInformation Theory*, 15(2), 306-315, Mar. 1969.

[4] J.C. Chang, Distance-increasing mappings from binary vectors to permuations. *IEEE Transactions on Information Theory*, 51(1): 359-363, Jan. 2005.

[5] J.C. Chang, Distance-increasing mappings from binary vectors to permutations that increase hamming distances by at least two. *IEEE Transactions on Information Theory*, 52(4): 1683-1689, Apr. 2006.

[6] J.C. Chang, R.J. Chen, T. Kløve and S.C. Tsai, Distance-preserving mappings from binary vectors to permutations. *IEEE Transactions on Information Theory*, 49(4):1054-1059, Apr. 2003.

[7] P. Diaconis, R. L. Graham, Spearman's Footrule as a Measure of Disarray. *Journal of the Royal Statistical Society*, 39(2): 262-, 1977.

[8] M. Deza, T. Huang, Metrics on Permutations, a Survey. *J. Comb. Inf. Sys. Sci.*, 23: 173-185. 1998

[9] P. Diaconis, Group Representations in Probability and Statistics. Institute of Mathematical Statistics, Hayward, California, 1988.

[10] J. Giesen,E. Schuberth,M. Stojakovic, Approximate sorting. *Theoretical Informatics, 7th Latin American Symposium*, 524-531, Mar. 2006.

[11] S. Haykin, *Communication Systems*, 4th Ed, John Wiley & Sons, Inc., USA.

[12] Hendrik C. Ferreira, A.J. Han Vinck, Interference cancellation with permutation trellis codes. *IEEE VTS-Fall VTC 2000*, 52(5):2401-2407, Sep. 2000.

[13] W.C. Huffman, V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge, U.K.: Cambridge Univ. Press, 2003.

[14] Kendall, Maurice, Gibbons, Jean Dickinson, *Rank correlation methods*, London, U.K.: Edward Arnold, 1990.

[15] Donakd E. Knuth, *The Art of Computer Programming Volumn 3 : Sorting and Searching*, US : Addison Wesley Longman, second edition, 1998.

[16] K. Lee, New distance-preserving maps of odd length. *IEEE Transactions on Information Theory*, 50(10): 2539-2543, Oct. 2004.

[17] K. Lee, Cyclic constructions of distance-preserving maps. *IEEE Transactions on Information Theory*, 51(12): 4392-4396, Dec. 2005.

[18] K. Lee, Distance-increasing maps of all lengths by simple mapping algorithms. *IEEE Transactions on Information Theory*, 52(7): 3344-3348, Jul. 2006.

[19] J.H. van Lint, R.M. Wilson, *A Course in Combinatiorics*, Cambridge, U.K.: Cambridge Univ. Press, second edition, 2001.

[20] T.G. Swart, H.C. Ferreira, A Generalized Upper Bound and a Multilevel Construction for Distance-Preserving Mappings. *Information Theory, IEEE Transactions* , 52(8): 3685- 3695, Aug. 2006.

[21] K.W. Shum, Permutation coding and MFSK modulation for frequency selective channel *IEEE Personal, Indoor and Mobile Radio Communications*, 13(5): 2063-2066, Sept. 2002

[22] E. Stoll and L. Kurz, Suboptimum Rank Detection Procedures Using Rank Vector Codes. *Communications, IEEE Transactions*, 16(3): 402-410, Jun. 1968.

[23] A. J. H.Vinck and J.Häring, Coding and modulation for power-line communications, *Proc. Int. Symp. Power Line Communication* Limerick, Ireland, April, 2000.

[24] A.J.H Vinck, J. Haering, T. Wadayama, Coded M-FSK for power line communications *Information Theory, Proceedings. IEEE International Symposium*, 2000, p.137.

[25] A. J. H. Vinck, Coded modulation for powerline communications. *Proc. Int. J. Electron. Commun*, 54(1): 45-49, 2000.