Decoding Permutation Arrays with Ternary Vectors

Chia-Jung Lee* Te-Tsung Lin † Min-Zheng Shieh‡ Shi-Chun Tsai ‡
Hsin-Lung Wu §

Abstract

We give an explicit decoding scheme for the permutation arrays under Hamming distance metric, where the encoding is constructed via a distance-preserving mapping from ternary vectors to permutations (**3-DPM**).

1 Introduction

In this paper we consider the set of permutations from $\{1, 2, \cdots, n\}$, denoted as S_n. We study the decoding issue with respect to an encoding scheme $C : \{0, 1, 2\}^n \to S_n$, i.e., given a corrupted permutation $\pi \in S_n$ that is close to a codeword $C(x) \in S_n$ we want to recover $x \in \{0, 1, 2\}^n$. We use the Hamming distance metric to measure the distance for ternary vectors and permutations. That is, for any $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n) \in \{0, 1, 2\}^n$, the distance between x and y is defined as $d_H(x, y) = |\{i : x_i \neq y_i\}|$, which is the number of positions they differ. Similarly, for π and $\tau \in S_n$, $d_H(\pi, \tau) = |\{i : \pi_i \neq \tau_i\}|$. An (n, d) permutation array (PA) is a subset of S_n with the property that the distance between any two permutations in the array is at least d. PAs have been studied for some time, see e.g. [1] for a list of early references. Vinck [8, 7] proposed permutation arrays as an error correcting code over power-line communications, where each symbol $i \in \{1, \ldots, n\}$ is associated with a frequency f_i and a message is encoded as a permutation. Most of the previous works have been on finding mappings from binary vectors to permutations that preserve the minimum distance of the binary vectors. A typical approach starts by encoding a message with a binary code, which is then mapped to a permutation and transmitted. From a correctly received permutation, one can find the corresponding binary vector, with which one can recover the message. But, there is little discussion on correcting errors directly from the permutations when the error is not erasure. It was not clear how to efficiently recover the message, when the received permutation was not a codeword. Using l_∞-norm to measure the distance between permutations, Lin et al. [5, 3] proposed the first efficient encoding and decoding scheme for permutation arrays. In this paper, we investigate the same problem but with the Hamming distance metric, which is widely used in coding theory and related areas[2]. Meanwhile, Lin et al. [6] gave a simple distance-preserving

*Email: leecj@iis.sinica.edu.tw, this work was done while the author was with National Chiao Tung University.
†Email: atman.cs94g@nctu.edu.tw, this work was done while the author was with National Chiao Tung University
‡Email:{mzhsieh, sctsai}@csie.nctu.edu.tw, Department of Computer Science, National Chiao Tung University, Taiwan.
§Email: hsinlung@mail.ntpu.edu.tw, this work was done while the author was with National Chiao Tung University.
The work was supported in part by the National Science Council of Taiwan under contract NSC 97-2221-E-009-064-MY3 and and NSC-98-2221-E-009-078-MY3.

mapping from ternary vectors into permutations. Based on their construction from $\{0,1,2\}^n$ to S_n for $n \geq 16$, we develop an efficient decoding scheme for permutation arrays. For the rest of the paper, we review the encoding algorithms in section 2 and show our main result, the decoding algorithm, in Section 3.

2 Construction of 3-DPM

For completeness, we include the construction of ternary distance preserving mapping (**3-DPM**)[5]. First of all, we show the algorithm for input length $8m$ for any integer $m \geq 2$. We call the algorithm A_{8m}. Then A_{8m} can be extended to all input of length at least 16. The approach gives a framework for designing general q-DPM. For convenience, we use the notation $[n]$ to denote the set $\{1,2,\ldots,n\}$ and $[i,j]$ for the set $\{i,i+1,\ldots,j\}$, where i and j are integers and $i<j$.

2.1 3-DPM of length $8m$ for $m \geq 2$

The 3-DPM of length $8m$ (A_{8m}) is shown in Figure 1. Algorithm A_{8m} consists of two passes: PASS 1 and PASS 2. The encoding scheme for $n = 16$ is illustrated in Figure 2, which is similar to the sorting network[4]. Each vertical line segment connects two horizontal lines and the value of the variable next to it determines to swap the values on the horizontal lines or not. During PASS 1, if the variable associated with a vertical line segment has value 1, then the values of the two corresponding horizontal lines are swapped. For PASS 2, the values will be swapped when the associated variable has value 2. We summarize the transitions of PASS 1 and 2 in Table 1 and 2. Table 2 is very similar to Table 1 if we replace 1 by 2. Note that addition and substraction are operated in $Z_{8m} = [8m]$, that is, if $a, b \in Z_{8m}$ then the output of $a+b$ is $(a+b \bmod 8m)$ if $(a+b \bmod 8m) \neq 0$; $8m$ otherwise.

Table 1: Possible values of π_j^1 after PASS 1 for $k \in \{0,1,\ldots,4m-1\}$.

	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	$2k+4$	$2k+1$

2.2 3-DPM for input length ≥ 16

Algorithm A_{8m} has a nice property as shown in lemma 1 and can be extended to handle input length greater than or equal to 16. As in the previous subsection, let $\pi = A_{8m}(x)$ and π^1 be the intermediate result after PASS 1. The 3-DPM algorithm A_{8m+k} is shown in Figure 3.

Algorithm A_{8m}:
Input: $(x_1, \ldots, x_{8m}) \in Z_3^{8m}$
Output: $(\pi_1, \ldots, \pi_{8m}) \in S_{8m}$

PASS 1 :
$(\pi_1^1, \pi_2^1, \ldots, \pi_{8m}^1) \leftarrow (1, 2, \ldots, 8m)$;
for $i = 0$ **to** $4m - 1$ **do**;
    if $x_{2i+1} = 1$ **then swap** $(\pi_{2i+1}^1, \pi_{2i+2}^1)$;
for $i = 0$ **to** $4m - 1$ **do**;
    if $x_{2i+2} = 1$ **then swap** $(\pi_{2i+2}^1, \pi_{2i+3}^1)$;

PASS 2 :
$(\pi_1, \pi_2, \ldots, \pi_{8m}) \leftarrow (\pi_1^1, \pi_2^1, \ldots, \pi_{8m}^1)$;
for $i = 0$ **to** $m - 1$ **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** $m - 1$ **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 $(\pi_1, \ldots, \pi_{8m})$.

Figure 1: 3-DPM Algorithm A_{8m}

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

	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}

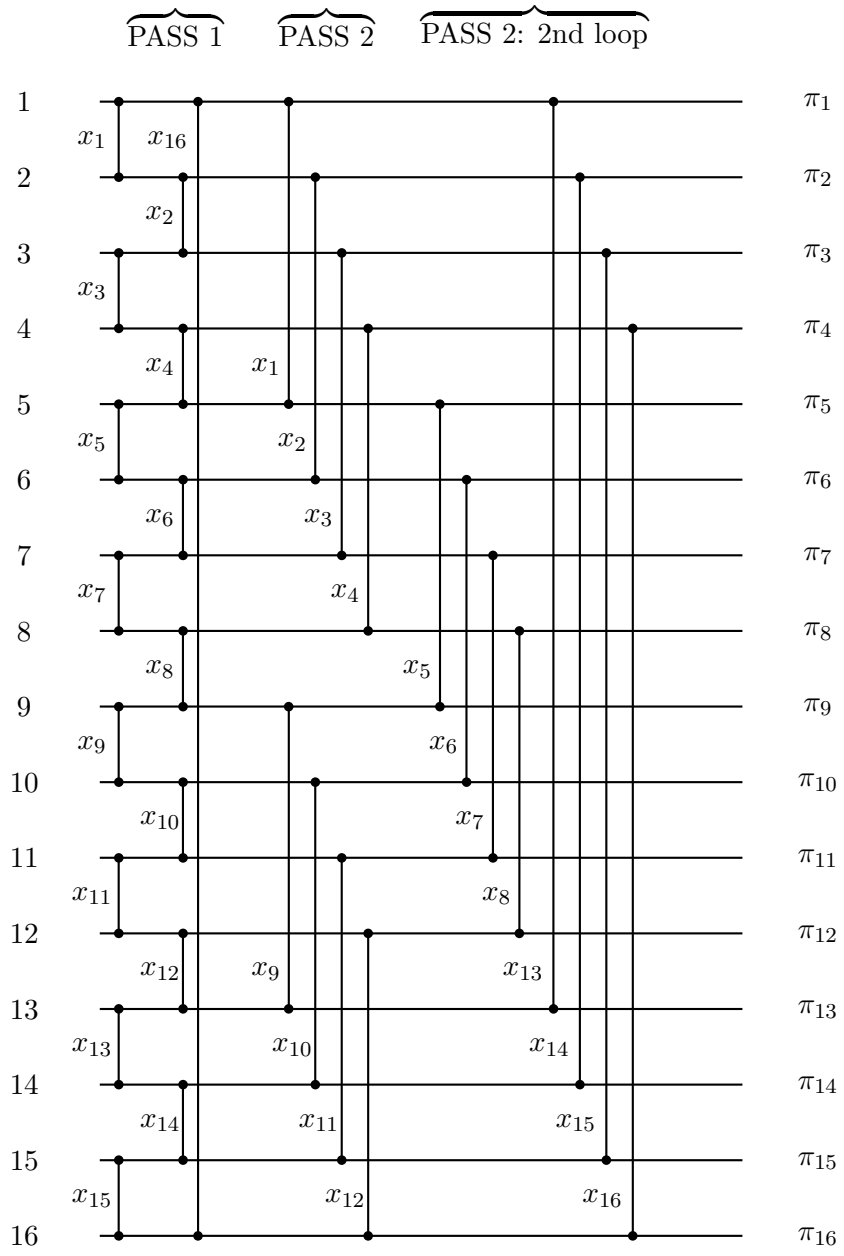

Figure 2: Encoding with sorting network for $n = 16$. Initially, line i has value i. After PASS 1, line i has value π_i^1. Each vertical line segment has an associated variable to its left, which decides to swap the values of two horizontal lines connecting to it or not.

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

Figure 3: 3-DPM Algorithm A_{8m+k} for $k \in \{1, \ldots, 7\}$

Lemma 1. *[6] For any $i \in \{1, 2, \cdots, 8m\}$, $\pi_i \neq i - 3$.*

Theorem 1. *[6] $A_{8m+k} : Z_3^{8m+k} \to S_{8m+k}$ is a* **3-DPM** *for all $m \geq 2$ and $k \in \{1, \cdots, 7\}$.*

Corollary 1. *[6] There exists an explicit construction of 3-DPM 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.

3 Decoding Permutation Arrays with 3-DPM

As shown in [1], we can use 3-DPM to construct permutation arrays with Hamming distance. In this section we show the corresponding decoding algorithm.

Suppose C is an (n, d) ternary code, which can correct up to e errors. Let $n = 8m+k$, $m \geq 2$ and $0 \leq k \leq 7$. By Theorem 1 and $m \geq 2$, we have a distance-preserving mapping $A_{8m+k} : Z_3^n \to S_n$. It is easy to see that $A_{8m+k}(C)$ is a permutation array of length n with minimum distance d. Let P be $A_{8m+k}(C)$ and so $|P| = |A_{8m+k}(C)| = |C|$.

We show the encoding/decoding scheme in Figure 4. For the encoding issue, if C has an efficient encoding algorithm $E : \text{Msg} \to Z_3^n$, where Msg is an arbitrary message space with size equal to $|C|$. Let $E_P = A_{8m+k} \circ E$, then $E_P : \text{Msg} \to S_n$ is an efficient encoding algorithm for P because E and A_{8m+k} are both efficient.

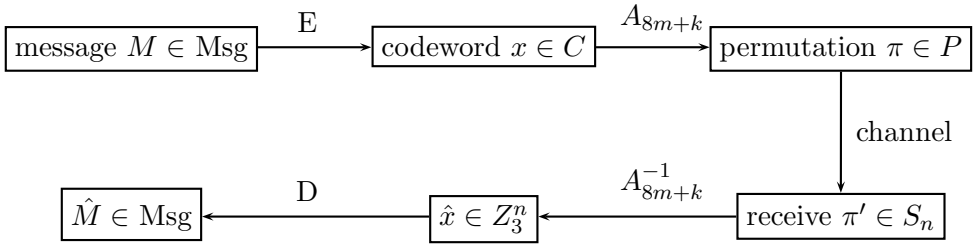

Figure 4: Encoding and decoding with permutation arrays and 3-DPM

Now 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 $\pi = A_{8m+k}(x) \in P$ and π' be a corrupted permutation satisfying $d_H(\pi, \pi') = d$. Without decoding π' to π directly, we design an algorithm A^{-1}_{8m+k} which computes the inversion of A_{8m+k}. If we can bound $d_H(A^{-1}_{8m+k}(\pi'), x)$ by $d_H(\pi, \pi')$, then we can decode P by combining A^{-1}_{8m+k} and D.

To understand the decoding algorithm, let us explain the idea first. Note that A_{8m+k} is based on A_{8m} and then handle the last k positions. We consider the inversion of A_{8m} first. The idea is based on the proof of lemma 1[6], which shows for any i, $\pi_i \neq i-3$ by checking the path of symbol $i-3$ and deriving a contradiction on the value of x_{i-4}. In general if $\pi_i = t$, we can determine the transition 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 and 4, where each π_i can have up to 15 legitimate values.

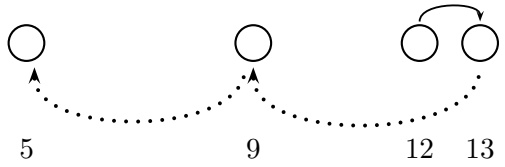

Figure 5: The unique path of symbol 12, given $\pi_5 = 12$.

Tables 3 and 4 show each possible value of π_i. For example, suppose $\pi_i = i + 7$, (i mod 8) $\in \{5, 6, 7, 8\}$ and i is odd(e.g. the case $\pi_5 = 12$). In the gray area of Table 4, it implies that $\pi_i = \pi^1_{i+8}$, $\pi^1_{i+8} = i + 7$ and gives 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 both tables by checking algorithm A_{8m}. Note that some entries are not applicable(n.a.), since A_{8m} avoids some values for certain positions in a permutation.

Checking each position of π, we can determine all the values of x_i's and compute the inversion of A_{8m}. Now consider A^{-1}_{8m+k}. If $\pi_i = t$ where $i \in \{8m+1, \ldots, 8m+k\}$, then we set $x_i = 0$ if $\pi_i = i$, $x_i = 1$ if $\pi_i = i-3$, and $x_i = 2$ otherwise. If $\pi_i = t$ with $i \in \{1, \ldots, 8m\}$ and $t \in \{8m+1, \ldots, 8m+k\}$, it implies π_i must have been swapped with $\pi_t = t$ in the final stage of algorithm A_{8m+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}_{8m+k} as follows.

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

$i = 8k + 8 + j$, $j \in \{1, 2, 3, 4\}$, $i \in [n]$, $k \in \{0, \ldots, \lfloor \frac{n}{8} \rfloor - 1\}$				
	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 4: Path Table of π_i , $i = 8k + 4 + j$, $j \in \{1, 2, 3, 4\}$

$i = 8k + 4 + j$, $j \in \{1, 2, 3, 4\}$, $i \in [n]$, $k \in \{0, \ldots, \lfloor \frac{n}{8} \rfloor - 1\}$				
	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} $x_i = 2$, $x_{i+4} = 2$	$i+6$	$x_{i+6} = 1$, $x_{i+7} = 1$	$i+7$	$x_{i+7} = 1$, $x_{i+8} \neq 1$
	$i+7$	$x_{i+6} \neq 1$, $x_{i+7} = 1$	$i+8$	$x_{i+7} \neq 1$, $x_{i+8} \neq 1$
	$i+8$	$x_{i+7} \neq 1$, $x_{i+8} \neq 1$	$i+9$	$x_{i+8} = 1$, $x_{i+9} \neq 1$
	$i+9$	$x_{i+7} \neq 1$, $x_{i+8} = 1$	$i+10$	$x_{i+8} = 1$, $x_{i+9} = 1$

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

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

(2) For all i in $\{1,\ldots,8m\}$, if π_i is larger than $8m$, then swap (π_i, π_{π_i}).

(3) For each π_i, $i \in \{1,\ldots,8m\}$, let B_i be an empty bucket for x_i. By the value of i and π_i, find the corresponding entries in Table 3 or Table 4. If it is not in the tables or not applicable(n.a), then do nothing. Else determine the values of four positions of x, from which obtain $x_i = b$ by checking the tables, put b to B_i if $b \in \{1,2\}$. If $b \notin \{1,2\}$, then put 0 to B_i.

(4) Determine x_i by a weighted majority vote. In each bucket, '0' has weight 0.5; '1' and '2' each has weight 1. For each i in $\{1,\ldots,8m\}$, check B_i, assign x_i to be the value $b \in \{0,1,2\}$ with the largest weight. If tie, choose the larger value.

First we use π_{8m+i} to decide x_{8m+i} for all $i \in \{1,\ldots,k\}$. One can verify that if π_{8m+k} is not corrupted then x_{8m+k} is correct. Next for $i \in \{1,\ldots,8m\}$, if $\pi_i > 8m$, it implies A_{8m+k} swaps π_i and π_{π_i}, and then we should swap them. Third, bucket B_i is used to collect the votes (information) of x_i. For each $\pi_i = t$, one can determine the values of x in four positions by checking Table 3 and Table 4. If $x_i = b \in \{1,2\}$ then put b to B_i; if $x_i \notin \{1,2\}$ then put '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 determine the value of x_i. Because if it gives $x_i \neq 1$(or 2) then we put 0 to the 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. Note that Step (3) also indicates that Algorithm A^{-1}_{8m+k} can handle erasure error.

Each π'_i contributes information in at most 4 positions of x. For each x_i, four positions of π' contribute correct information of x_i, if π' is not corrupted, since each x_i is used to decide whether to swap two positions or not in PASS 1, likewise in PASS 2. Thus it reveals some information about x_i by checking the path of those four symbols.

The inverse algorithm A^{-1}_{8m+k} works well if π is not corrupted. Let us consider the corrupted π'. By Tables 3 and 4, each error will give wrong information in at most 4 positions of x, and also lose correct information in at most 4 positions of x. It immediately gives a rough bound $d_H(A^{-1}_{8m+k}(\pi'), x) \leq 8 \cdot d_H(\pi, \pi')$. Here we give a better bound by analyzing it more carefully. Let $\pi = A_{8m+k}(x)$ be the correct codeword of x, and π' a corrupted permutation.

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

Proof. Let $x' = A^{-1}_{8m+k}(\pi')$. First note $|B_i| = 4$ for all $i \in \{1,\ldots,8m\}$ if π' is not corrupted by Table 3 and Table 4. 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 from two corrupted π'_i's. Each π'_i votes in at most four buckets and it causes at most eight changes on the

buckets, since a wrong vote can have two effects, i.e., removing a vote from one bucket and adding an extra vote to another.

Now we estimate the total effects of the corrupted positions. Let $d_H(\pi_{[1,8m]}, \pi'_{[1,8m]}) = d_1$, $d_H(\pi_{[8m+1,8m+k]}, \pi'_{[8m+1,8m+k]}) = d_2$, and $d = d_1+d_2$. It's clear $d_H(x_{[8m+1,8m+k]}, x'_{[8m+1,8m+k]}) \leq d_2$ because $\pi_i = \pi'_i$ implies $x_i = x'_i$ for $i \in [8m+1, 8m+k]$. For each $\pi_i \neq \pi'_i$ with $i \in [8m]$, by the above observation, it causes at most $8/2 = 4$ wrong decisions on average in the weighted majority vote. For each $i \in [8m]$ with $\pi_i = \pi'_i$, if $\pi'_i > 8m$, 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 $d_H(x_{[1,8m]}, x'_{[1,8m]}) \leq 4 * (d_1 + d_2) = 4d$. Finally, we have $d_H(x, x') = d_H(x_{[1,8m]}, x'_{[1,8m]}) + d_H(x_{[8m+1,8m+k]}, x'_{[8m+1,8m+k]}) \leq 4d + k \leq 4 \cdot d_H(\pi, \pi') + k$.

□

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

Theorem 2. *For all $n \geq 16$ and $d \leq n$, suppose C is an (n, d) ternary code and P is an (n, d) permutation array generated by A_{8m+k}. 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' satisfies $d_H(\pi, \pi') \leq e/4 - 2$, for some codeword $\pi \in P$.*

Note that the decoding scheme will also work when the received corrupted codeword $y \in \{1, 2, \cdots, n\}^n$ is not a permutation but satisfies $d_H(\pi, y) \leq e/4 - 2$, 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 when the error does not exceed $e/4$ too much. This leads us to conjecture that the bound $e/4$ may not be tight.

References

[1] 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.

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

[3] T. Kløve, T.-T. Lin, S.-C. Tsai and W,-G. Tzeng, Permutation arrays under the Chebyshev distance, *IEEE Transactions on Information Theory*, 56(6): 2611–2617, June 2010.

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

[5] T.-T. Lin, S.-C. Tsai,W.-G. Tzeng, Efficient Encoding and Decoding with Permutation Arrays, *Proceedings 2008 IEEE International Symposium on Information Theory*, Toronto, Ontario, Canada, July 2008, pp. 211-214.

[6] T.-T. Lin, S.-C. Tsai, H.-L. Wu, Simple Distance-Preserving Mapping from Ternary Vectors to Permutations. *IEEE Transactions on Information Theory*, 54(7):3251-3256, Jul. 2008.

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

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