Efficient Decoding of Permutation Codes Obtained from Distance Preserving Maps

Yeow Meng Chee and Punarbasu Purkayastha
Division of Mathematical Sciences, School of Physical & Mathematical Sciences
Nanyang Technological University, Singapore 637371
Emails: {ymchee, punarbasu}@ntu.edu.sg

Abstract—**We study the decoding of permutation codes obtained from distance preserving maps and distance increasing maps from Hamming space. We provide efficient algorithms for estimating the q-ary digits of the Hamming space so that decoding can be performed in the Hamming space.**

Index Terms—**Permutation codes, Distance preserving maps**

I. Introduction

Transmission of data over high voltage electric power lines is a challenging problem due to the harsh nature of the channel. The noise characteristics of this channel, called the powerline communication (PLC) channel, include permanent narrowband noise, impulse noise, in addition to fading and additive white Guassian noise. Vinck [8] studied this channel and showed that M-ary Frequency Shift Keying (M-FSK) modulation, in conjunction with the use of permutation codes, provide the required redundancy to correct the errors resulting from the harsh noise pattern. This has given rise to increased research on codes in the permutation space (see [4] for a survey). One method to obtain a permutation code is to consider distance increasing maps (DIMs) or distance preserving maps (DPMs) from the Hamming space to the permutation space. The works in [3], [5]–[7] address the problem of constructing such DIMs and DPMs. Unlike the case of codes in the Hamming space, decoding of codes in the permutation space is a more difficult problem, especially because of the loss of linearity. Bailey [1] gave efficient decoding algorithms in the case when the permutation codes are subgroups. Unfortunately, the permutation codes obtained from DIMs or DPMs of codes in the Hamming space are not permutation groups in general. Swart and Ferreira [7] studied some DIMs and DPMs from the binary Hamming space to permutations and provided efficient decoding algorithms for determining the binary vectors.

In this work we study the problem of decoding permutation codes obtained from DIMs or DPMs of q-ary Hamming codes. The main idea that we employ is to perform only *estimation* of the q-ary digits from the received vector. The actual decoding of the estimated q-ary vector is performed in the q-ary Hamming space. Decoding of linear codes is a very well studied problem and many efficient decoding algorithms exist. Our aim here is to provide efficient ways of estimating the q-ary digits so that the overall estimation and decoding procedure still retains low complexity.

We use the notation S_N to denote the permutation space over the symbols $\{0, \dots, N-1\}$. Each element σ of S_N is written as a vector $\sigma = (\sigma_0, \dots, \sigma_{N-1})$, which represents the output of the permutation. The distance between two elements of S_N is taken to be the Hamming distance between their vector representations. We use the notation $\mathbb{Z}_q^n = \{0, \dots, q-1\}^n$ to denote the Hamming space. A *distance-preserving map* $\Pi : \mathbb{Z}_q^n \to S_N$ is a mapping which preserves the Hamming distance between any two vectors, that is, $d(\Pi(\boldsymbol{x}), \Pi(\boldsymbol{y})) \geq d(\boldsymbol{x}, \boldsymbol{y})$ for any vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_q^n$. A *distance-increasing map* $\Pi : \mathbb{Z}_q^n \to S_N$ strictly increases the Hamming distance, that is, $d(\Pi(\boldsymbol{x}), \Pi(\boldsymbol{y})) > d(\boldsymbol{x}, \boldsymbol{y})$ for any distinct vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_q^n$.

An upper bound on the size of a permutation code with minimum distance d is given by $N!/(d-1)!$. Clearly, the information rate of a permutation code can be larger than the rate achievable by DPMs from q-ary Hamming space (unless q is proportional to N). Sharply k-transitive groups, which have efficient decoding, are known to achieve this upper bound (see [1], [2]). For such groups either $d \leq 3$ or $d \geq N-4$. In the latter case, the size of the code is only polynomial in N. Thus it is of interest to consider other means of obtaining permutation codes, for instance from DPMs.

In the following sections we consider very specific DIMs and DPMs. All the DIMs and DPMs we use are non-length preserving, but ensure that efficient estimation algorithms exist. Hence, the rate of the codes decreases by a factor of $1/\lceil \log_2 q \rceil$, compared to the q-ary code in the Hamming space. We consider a channel, for instance the PLC channel, which introduces both errors and erasures. The simplest such algorithm, and DIM from the binary Hamming space, introduced in the next section has only linear complexity. This algorithm also guarantees that the estimation procedure does not introduce extra errors or erasures in the binary digits. The mappings in the subsequent sections are more complicated and require at least two symbols in the permutation space to estimate one q-ary digit. Hence, such guarantees can be provided if the channel introduces only erasures.

II. DIM from binary vectors to permutations

In this section we discuss a DIM from binary vectors to permutations. Lee [5] studied a DIM and its properties, which is similar to this DIM (also, [7, Eg. 1]). We give an efficient algorithm in the permutation space which provides only an *estimate* of the bits. We first describe the DIM used here.

The DIM maps a binary vector $\boldsymbol{b} = (b_0, \dots, b_{n-1})$ of length n to a vector $\sigma = (\sigma_0, \dots, \sigma_n)$ of length $n+1$ in S_{n+1}.

We start from the identity permutation $\sigma^{(-1)} = (0, \ldots, n)$. The bit b_0 permutes the first two coordinates, resulting in a vector $\sigma^{(0)} = (\sigma_0^{(0)}, \ldots, \sigma_n^{(0)})$. For $i = 1, \ldots, n-1$ the bit b_i permutes the coordinates $\sigma_i^{(i-1)}$ and $\sigma_{i+1}^{(i-1)}$ of $\sigma^{(i-1)}$. Let Π_0 denote this mapping. The example below illustrates the map of $\boldsymbol{b} = (1, 1, 0, 1)$ to the permutation vector $(1, 2, 0, 4, 3)$. For brevity of exposition we write the vector σ in a compact form, $\sigma = 12043$. Underlined portions denote the affected symbols.

EXAMPLE *2.1:* $01234 \xrightarrow{b_0=1} \underline{10}234 \xrightarrow{b_1=1} 1\underline{20}34 \xrightarrow{b_2=0} 12\underline{03}4 \xrightarrow{b_3=1} 120\underline{43}$.

ALGORITHM *2.2:* DIM Π_0 from $\mathbb{Z}_2^n$ to S_{n+1}

```
**Input**: $\boldsymbol{b} = (b_0, \ldots, b_{n-1}) \in \mathbb{Z}_2^n$
**Output**: $\sigma = (\sigma_0, \ldots, \sigma_n) \in S_{n+1}$
$\sigma^{(-1)} \leftarrow (0, \ldots, n)$
**for** $i$ from 0 to $n-1$
    $\sigma^{(i)} \leftarrow \sigma^{(i-1)}$
    **if** $b_i = 1$ then
        $\sigma_i^{(i)} \leftarrow \sigma_{i+1}^{(i-1)}, \quad \sigma_{i+1}^{(i)} \leftarrow \sigma_i^{(i-1)}$
```

The proposition below proves that the mapping is indeed a DIM.

Proposition 2.3: (see [5]) The mapping Π_0 is a DIM with $d(\Pi_0(\boldsymbol{b}), \Pi_0(\boldsymbol{b}')) \geq d(\boldsymbol{b}, \boldsymbol{b}') + n_R$, where n_R is the number of runs of ones in $\operatorname{supp} \boldsymbol{b} \cup \operatorname{supp} \boldsymbol{b}'$, where $\operatorname{supp} \boldsymbol{b}$ denotes the support of the vector $\boldsymbol{b}$.

Proof: We consider three cases that the supports of the two vectors $\boldsymbol{b}$ and $\boldsymbol{b}'$ can satisfy. First, suppose that $\boldsymbol{b}$ contains the following segment in the first l coordinates (WLOG, we may assume it is the first l coordinates)

$$\underbrace{1, 1, \ldots, 1}_{l},$$

and $\boldsymbol{b}'$ contains the segment in the first l coordinates

$$\underbrace{0, \ldots, 0}_{m_1} \underbrace{1, \ldots, 1}_{m}, \underbrace{0, \ldots, 0}_{m_2},$$

where $m_1 + m + m_2 = l$ with $0 < m < l$ and $m_1, m_2 \geq 0$. Then the first l coordinates of $\Pi_0(\boldsymbol{b})$ is $(1, 2, \ldots, l)$, and the first l coordinates of $\Pi_0(\boldsymbol{b}')$ is $(0, 1, \ldots, m_1 - 1, m_1 + 1, \ldots, m_1 + m, m_1, m_1 + m + 1, \ldots, l - 1)$. Thus, the proposition is satisfied in the first l coordinates.

The second case that may arise is of the following form. Let, WLOG, the first l coordinates of $\boldsymbol{b}$ be

$$\underbrace{0, \ldots, 0}_{m} \underbrace{1, \ldots, 1}_{l-m},$$

where $l > m > 0$. Let the first l coordinates of $\boldsymbol{b}'$ be of the form

$$\underbrace{1, \ldots, 1}_{m}, \underbrace{1, \ldots, 1}_{k}, \underbrace{0, \ldots, 0}_{l-m-k},$$

where $m, k > 0$ and $l > m + k$. Then the first l coordinates of $\Pi_0(\boldsymbol{b})$ is of the form $(0, 1, \ldots, m-1, m+1, \ldots, l)$ and the first l coordinates of $\Pi_0(\boldsymbol{b}')$ is $(1, 2, \ldots, m+k, 0, m+k+1, \ldots, l-1)$. Thus the proposition is satisfied since the number of runs of ones in $\operatorname{supp} \boldsymbol{b} \cup \operatorname{supp} \boldsymbol{b}'$ is one in the first l coordinates.

Finally, the third case is of the form where the first l coordinates of $\boldsymbol{b}$ is $(0, \ldots, 0)$ coordinates but the first l coordinates of $\boldsymbol{b}'$ is $(1, \ldots, 1, 0)$. Then the first l coordinates of $\Pi_0(\boldsymbol{b})$ contains the vector $(0, 1, \ldots, l-1)$ while the first l coordinates of $\Pi_0(\boldsymbol{b}')$ is $(1, 2 \ldots, l-1, 0)$. Thus, in all the cases the distance increases by the number of runs of ones in $\operatorname{supp} \boldsymbol{b} \cup \operatorname{supp} \boldsymbol{b}'$. ■ ■

A. Estimating bits from the permutation vector

A very simple estimation procedure gives the correct binary bit if the received symbol is correct in the corresponding coordinate. The algorithm is described below. We denote an erasure by the symbol ε. Let the vector received as the output of the channel be denoted by $\boldsymbol{y}$. It lies in the space $\{\mathbb{Z}_{n+1} \cup \varepsilon\}^{n+1}$.

ALGORITHM *2.4:* Estimating bits from $\boldsymbol{y}$

```
**Input**: $\boldsymbol{y} = (y_0, \ldots, y_n) \in \{\mathbb{Z}_{n+1} \cup \varepsilon\}^{n+1}$
**Output**: $\hat{\boldsymbol{b}} = (\hat{b}_0, \ldots, \hat{b}_{n-1}) \in \{\mathbb{Z}_2 \cup \varepsilon\}^n$
**for** $i$ from 0 to $n-1$
    **if** $y_i = i + 1$ then $\hat{b}_i \leftarrow 1$
    **elseif** $y_i < i + 1$ then $\hat{b}_i \leftarrow 0$
    **else** $\hat{b}_i \leftarrow \varepsilon$.
```

The estimate $\hat{\boldsymbol{b}}$ can be now provided to the decoder for the binary code for decoding. Clearly, the above algorithm never introduces any error if the coordinate y_i is correct. Hence, this procedure can correctly decode with a bounded distance decoder if the number of errors n_e and the number of erasures n_ε satisfy the condition $2n_e + n_\varepsilon < d$, where d is the minimum distance of the binary code. In practice the algorithm potentially corrects more errors. For example if the transmitted symbol corresponding to $b_i = 0$ is different from the received symbol y_i, but the received symbol satisfies $y_i < i + 1$ then there is no error in estimating the bit $\hat{b}_i$. This algorithm performs at most $2n$ comparisons and has a memory requirement of exactly one symbol at each step. In comparison the decoding algorithm in Swart & Ferreira [7] performs decoding in the permutation space, under M-FSK signaling, and requires $O(M^2 + nM)$ computations and $o(3^{nM^2})$ memory.

If the rate of the binary code is R then the rate of transmission of information bits is $Rn/(n+1)$. From the DIM and the estimation algorithm it can be inferred that only about "half" the permutation space is used for communication over an M-FSK channel. At the i-th time instance, the symbols $i + 2, \ldots, n$ are used neither during transmission nor during the decoding procedure. If the DIM is from a linear binary code of dimension $k = Rn$, then one can achieve a rate of $k/(n+1) + (k-3)/(n-2)$ by utilizing the unused symbols to transmit a shortened codeword $\tilde{\boldsymbol{b}}$ of length $n - 3$, but in *reverse order* of the DIM. If $\tilde{\sigma}^{(-1)} = (3, \ldots, n)$ then $\tilde{b}_0$ flips $\tilde{\sigma}_{n-3}^{(-1)}$ and $\tilde{\sigma}_{n-4}^{(-1)}$, $\tilde{b}_1$ flips $\tilde{\sigma}_{n-4}^{(0)}$, $\tilde{\sigma}_{n-5}^{(0)}$, and so on.

III. DIM FROM 2^m-ARY VECTORS TO PERMUTATIONS

In this section we describe a modification to the mapping in Section II so that it can be used for q-ary vectors where $q = 2^m$. The primary aim is to provide a simple means of estimating the symbols used. The idea is to use a binary representation of each symbol and map that binary representation of length m to an $m+1$ length permutation vector. We give an example below and then we describe the algorithm formally. We denote this mapping by Π_1. For brevity, we write the vectors in a compact form.

EXAMPLE *3.1:* Let $q = 2^2$ and let the symbols $\{0,1,2,3\}$ be mapped to their natural binary representation as $0 \mapsto 00, 1 \mapsto 01, 2 \mapsto 10$, and $3 \mapsto 11$. The vector $\boldsymbol{s} = 132$ is mapped to the permutation vector 0234516 in the following sequence of steps: $0123456 \xrightarrow{01} \underline{021}3456 \xrightarrow{11} 02\underline{341}56 \xrightarrow{10} 0234\underline{516}$. The underlined portions denote the affected symbols.

ALGORITHM *3.2:* DIM Π_1 from $\mathbb{Z}_{2^m}^n$ to S_{mn+1}

```
Input: $\boldsymbol{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_{2^m}^n$
Output: $\sigma = (\sigma_0, \ldots, \sigma_{mn}) \in S_{mn+1}$
$\sigma^{(-1)} \leftarrow (0, 1, \ldots, mn)$
for $i$ from 0 to $n-1$
    for $j$ from 0 to $m-1$
        $\sigma^{(im+j)} \leftarrow \sigma^{(im+j-1)}$
        $\boldsymbol{b}_i = (b_{i,0}, \ldots, b_{i,m-1})$, binary representation of $s_i$
        if $b_{i,j} = 1$ then
            $\sigma_{im+j+1}^{(im+j)} \leftarrow \sigma_{im+j}^{(im+j-1)}$
            $\sigma_{im+j}^{(im+j)} \leftarrow \sigma_{im+j+1}^{(im+j-1)}$
```

The estimation procedure for the symbols is the same as described in Section II-A. We estimate the bits and then recombine the bits to form the symbols. This is an efficient and reasonably effective method of estimating the symbols provided that the number of errors and erasures introduced by the channel is low. The number of comparisons required is at most $2mn$. One drawback of this DIM is that the rate of transmission of information bits decreases by a factor of $\frac{1}{m}$.

IV. DPM FROM BINARY VECTORS TO PERMUTATIONS

In this section we develop a new distance preserving map (DPM) from binary vectors to permutation vectors, which allows us to estimate the binary symbols efficiently. The mapping converts a length n binary vector to a length $n+1$ permutation vector in S_{n+1}. This method is introduced so that it can be generalized to a new DPM from ternary vectors to permutations in the next section. The following lemma is essential to the constructions in the remaining sections.

Lemma 4.1: Let $(\sigma_0, \ldots, \sigma_l)$ be a permutation of $(0, 1, \ldots, l)$. Then $\sigma = (\sigma_0 + i, \ldots, \sigma_l + i, l+1+i, \ldots, l+j+i) \bmod (l+j+1)$ is a permutation of the vector $(0, 1, \ldots, l+j)$, and the modulo is performed on each coordinate of the vector.

Proof: Consider the vector $\Sigma = (0, \ldots, l, l+1, \ldots, l+j)$ in S_{l+j+1}. Adding i modulo $l+j+1$ to each coordinate of Σ results in a vector $\Sigma + i$ which is a cyclic shift of Σ to the left by i positions. Hence $\Sigma + i$ is also in S_{l+j+1}. Considered as an unordered tuple, the elements of σ are the same as the elements of $\Sigma + i$ and hence σ is also a vector in S_{l+j+1}. ■

We now describe the algorithm to map the binary vectors to the permutation vectors. For a binary vector $\boldsymbol{b} = (b_0, \ldots, b_{n-1})$ the algorithm proceeds recursively as follows. Consider the binary vector as a $\{0,1\}$-vector in $\mathbb{R}$. The algorithm is initialized by starting with the identity permutation represented as $\sigma^{(-1)} = (0, 1, \ldots, n)$. For each $i = 0, \ldots, n-1$, the element b_i is added to the first $i+2$ positions of the permutation vector $\sigma^{(i-1)}$ modulo $(i+2)$, where $\sigma^{(i-1)}$ is the vector resulting from the previous iteration. Denote the DPM by Π_2. The example below illustrates the procedure.

EXAMPLE *4.2:* We map 1101 to 32140 as follows: $01234 \xrightarrow{b_0=1} \underline{10}234 \xrightarrow{b_1=1} \underline{210}34 \xrightarrow{b_2=0} \underline{2103}4 \xrightarrow{b_3=1} \underline{32140}$.

ALGORITHM *4.3:* DPM Π_2 from $\mathbb{Z}_2^n$ to S_{n+1}

```
Input: $\boldsymbol{b} = (b_0, \ldots, b_{n-1}) \in \mathbb{Z}_2^n$
Output: $\sigma = (\sigma_0, \ldots, \sigma_n) \in S_{n+1}$
$\sigma^{(-1)} \leftarrow (0, 1, \ldots, n)$
for $i$ from 0 to $n-1$
    $\sigma^{(i)} \leftarrow \sigma^{(i-1)}$
    for $j$ from 0 to $i+1$
        $\sigma_j^{(i)} \leftarrow \sigma_j^{(i-1)} + b_i \mod (i+2)$
```

Proposition 4.4: Π_2 is a DPM from $\mathbb{Z}_2^n$ to S_{n+1}, that is for $\boldsymbol{b}, \boldsymbol{b}' \in \mathbb{Z}_2^n$, $d(\Pi_2(\boldsymbol{b}), \Pi_2(\boldsymbol{b}')) \geq d(\boldsymbol{b}, \boldsymbol{b}')$.

Before providing the proof of the proposition we determine the output of the mapping Π_2 as a nonrecursive function of the input bits b_i, $i = 0, \ldots, n-1$. For brevity of the exposition, we introduce the notation $[a]_p$ to denote $a \bmod p$. Recall that the binary vector $\boldsymbol{b}$ is considered as a $\{0,1\}$-vector over $\mathbb{R}$.

Lemma 4.5: If $\Pi_2(\boldsymbol{b}) = \sigma = (\sigma_0, \ldots, \sigma_n)$, then

$$\begin{aligned} \sigma_0 &= b_0 + \cdots + b_{n-1}, \\ \sigma_l &= [l + b_{l-1}]_{l+1} + b_l + \cdots + b_{n-1}, \quad l = 1, \ldots, n. \end{aligned}$$

Proof: The output of the mapping Π_2 is given by

$$\begin{aligned} \sigma_0 &= [\cdots[[b_0]_2 + b_1]_3 + \cdots + b_{n-1}]_{n+1}, \\ \sigma_l &= [\cdots[[l + b_{l-1}]_{l+1} + b_l]_{l+2} + \cdots + b_{n-1}]_{n+1}, \; l > 0. \end{aligned}$$

For any $l = 1, \ldots, n$, we have that $[l + b_{l-1}]_{l+1} \leq l$ and hence $[l + b_{l-1}]_{l+1} + b_l \leq l+1$. This implies $[[l + b_{l-1}]_{l+1} + b_l]_{l+2} = [l + b_{l-1}]_{l+1} + b_l$, that is, we can remove the modulo operation. Similarly, the modulo operations by $l+3, \ldots, n+1$ can be removed. The same argument shows that σ_0 can be obtained by adding up the bits over $\mathbb{R}$. ■

Proof of Proposition 4.4: Let $\boldsymbol{b} = (b_0, \ldots, b_{n-1})$ and $\boldsymbol{b}' = (b_0', \ldots, b_{n-1}')$, be $\{0,1\}$-vectors over $\mathbb{R}$. The proof proceeds by induction on the number of consecutive bits that are different between the vectors $\boldsymbol{b}$ and $\boldsymbol{b}'$. Suppose $b_{i-1} \neq b_{i-1}'$. Then we show that either $\sigma_i \neq \sigma_i'$ or $\sigma_{i-1} \neq \sigma_{i-1}'$.

Let $\Delta_i = \sum_{l=i}^{n-1} b_l - b_l'$. Note that if $b_0 \neq b_0'$ then the vectors are clearly different in at least the first 2 positions. So, let $i \geq 2$. First, suppose that $\Delta_i = 0$. Clearly, $b_{i-1} \neq b_{i-1}'$ implies $\sigma_i \neq \sigma_i'$. Now, assume that $\Delta_i \neq 0$. If $\sigma_i = \sigma_i'$ then without loss of generality assume that $b_{i-1} = 0$ and $b_{i-1}' = 1$. Using Lemma 4.5 in the equation $\sigma_i = \sigma_i'$ leads to the condition $i = -\Delta_i$. We claim that $\sigma_{i-1} \neq \sigma_{i-1}'$. Suppose not. We get

$$[i-1+b_{i-2}]_i + \sum_{l \geq i-1} b_l = [i-1+b_{i-2}]_i + \sum_{l \geq i-1} b_l'. \quad (1)$$

We consider the different possibilities of b_{i-2} and b_{i-2}'. If $b_{i-2} = b_{i-2}'$ then (1) results in $\Delta_i = 1$, a contradiction to $i = -\Delta_i$. Similarly, one obtains contradictions for other values of b_{i-2} and b_{i-2}'.

Finally, we show by induction that if $b_{i+j} \neq b_{i+j}'$, $j = 0, \ldots, k-1$, then $\sigma_{i+j} \neq \sigma_{i+j}'$ for at least k terms of $j \in \{0, \ldots, k\}$. The case $k = 1$ is proved above. Assume it is true for any $k-1$ consecutive b_{i+j}'s. The only non-trivial case we need to consider is if $\sigma_i = \sigma_i'$ and $\sigma_{i+k} = \sigma_{i+k}'$. We claim this is not possible. Suppose $\sigma_{i+k} = \sigma_{i+k}'$. Then using $b_{l-1} - b_{l-1}' \in \{-1, 0, 1\}$, for $l = 1, \ldots, n$, we write

$$\sigma_l - \sigma_l' = -(b_{l-1} - b_{l-1}')l + \Delta_l.$$

Using $\sigma_{i+k} - \sigma_{i+k}' = -(b_{i+k-1} - b_{i+k-1}')(i+k) + \Delta_{i+k} = 0$, we get

$$\begin{aligned}\sigma_i - \sigma_i' = -(b_{i-1} - b_{i-1}')i + \Delta_i - \Delta_{i+k-1} \\ + (b_{i+k-1} - b_{i+k-1}')(i+k+1).\end{aligned}$$

Since $|-(b_{i-1} - b_{i-1}')i + \Delta_i - \Delta_{i+k-1}| \leq i+k-1$ and the last term of $\sigma_i - \sigma_i'$ is $\pm(i+k+1)$, we get $\sigma_i \neq \sigma_i'$. ■

A. Estimating the bits from the permutation vector

In this section we consider a method to estimate the bits from the permutation vectors, with low complexity. The estimated bits can then be provided to the decoder of the binary code for further processing. The main idea behind the estimation method is the following lemma.

Lemma 4.6: Let $\Pi_2(\boldsymbol{b}) = \sigma$. The difference between any two coordinates σ_i, σ_j for $j > i$ satisfies

$$\sigma_i - \sigma_j \begin{cases} > 0, & \text{if } b_{j-1} = 1, \\ < 0, & \text{if } b_{j-1} = 0. \end{cases}$$

Proof: We get,

$$\sigma_i - \sigma_j = [i + b_{i-1}]_{i+1} + \sum_{l=i}^{j-2} b_l + b_{j-1} - [j + b_{j-1}]_{j+1}.$$

The statements in the lemma follow from the observation that $[i + b_{i-1}]_{i+1} + \sum_{l=i}^{j-2} b_l \leq j-1$. ■

Let the received vector from the channel be denoted by $\boldsymbol{y} \in \{\mathbb{Z}_{n+1} \cup \varepsilon\}^{n+1}$. By Lemma 4.6, it is clear that the simplest estimation algorithm will consider a pair of distinct coordinates y_i, y_j and determine b_{j-1} from their difference. This can lead to erroneous estimation if either of the two coordinates are in error. However, correct estimation of b_{j-1} is guaranteed if both the coordinates are correct. If $y_j = \varepsilon$, then b_{j-1} can not be determined from y_j and we set $\hat{b}_{j-1} = \varepsilon$. Algorithm 4.7 below describes the procedure.

The term $t > 0$ in the algorithm corresponds to performing a majority vote for the estimate $\hat{b}_j$ for each $j = 0, \ldots, n-1$. Algorithm 4.7 requires at most $n(n+1)$ additions and subtractions, and $3n + \frac{n}{2}(n+1)$ comparisons. By restricting $|L_j|$ to at most a constant number, say ℓ, the number of additions and subtractions can be reduced to at most $2\ell n$, at the cost of less reliable estimate of the bits in the higher indices. If the number of errors and erasures are small then one can expect the above algorithm to perform well even for small $|L_j|$.

ALGORITHM *4.7:* Estimating the binary vector $\hat{\boldsymbol{b}}$ from $\boldsymbol{y}$

```
Input: $\boldsymbol{y} = (y_0, \ldots, y_n) \in \{\mathbb{Z}_{n+1} \cup \varepsilon\}^{n+1}$
Output: $\hat{\boldsymbol{b}} = (\hat{b}_0, \ldots, \hat{b}_{n-1}) \in \{\mathbb{Z}_2 \cup \varepsilon\}^n$
$L_0 \leftarrow \phi$, the empty set
for $j$ from 1 to $n$
    $L_j \leftarrow L_{j-1} \cup \{j-1 : y_{j-1} \neq \varepsilon\}$
    if $y_j = \varepsilon$ then $\hat{b}_{j-1} \leftarrow \varepsilon$
    else $t \leftarrow 0$
        for each $l$ in $L_j$
            if $y_l - y_j > 0$ then $t \leftarrow t+1$
            else $t \leftarrow t-1$
        if $t > 0$ then $\hat{b}_{j-1} \leftarrow 1$
        elseif $t < 0$ then $\hat{b}_{j-1} \leftarrow 0$
        else $\hat{b}_{j-1} \leftarrow \varepsilon$
```

B. Estimating bits on an erasure channel

The above algorithm simplifies significantly on an erasure channel. Using $|L_j| = 1$ is sufficient to guarantee that the number of erasures in the estimated bits $\hat{\boldsymbol{b}}$ is at most the number of erasures in the received symbols $\boldsymbol{y}$. In addition, if the symbol 0 of S_{n+1} is present in the received vector $\boldsymbol{y}$, then one can immediately estimate all the succeeding bits correctly, irrespective of the received values from the channel. This observation is formalized in the following lemma.

Lemma 4.8: Let $\boldsymbol{b} \in \mathbb{Z}_2^n$ and let $\sigma = \Pi_2(\boldsymbol{b})$. If $\sigma_j = 0$, then $b_{j-1} = 1$ and $b_l = 0$ for all $l \geq j$.

Proof: If $\sigma_j = 0$ then we have $[j + b_{j-1}]_{j+1} + b_j + \cdots + b_{n-1} = 0$ over the reals. This can be achieved only when $b_{j-1} = 1$ and $b_j = \cdots = b_{n-1} = 0$. ■

In the following section we describe how to extend the algorithms in this section to a DPM from ternary vectors to permutations.

V. DPM FROM TERNARY VECTORS TO PERMUTATIONS

Consider a DPM from ternary vectors in $\mathbb{Z}_3^n$ to permutation vectors in S_{2n+1}. For a ternary vector $\boldsymbol{s} = (s_0, \ldots, s_{n-1})$, the element s_i permutes the first $2i + 3$ coordinates of the permutation vector. As in the previous section, the construction is recursive and the final permutation vector also affords a nonrecursive representation in terms of the ternary digits. We describe the algorithm below. Let the mapping be denoted by Π_3 and consider the ternary digits as elements of the real field $\mathbb{R}$ for all the operations below. We first illustrate this algorithm by an example below.

EXAMPLE *5.1:* We map 121 to 4531260 using Π_3 as follows. $0123456 \xrightarrow{s_0=1} \underline{120}3456 \xrightarrow{s_1=2} \underline{34201}56 \xrightarrow{s_2=1} \underline{4531260}$.

ALGORITHM *5.2:* DPM Π_3 from $\mathbb{Z}_3^n$ to S_{2n+1}

```
Input: $\boldsymbol{s} = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_3^n$
Output: $\sigma = (\sigma_0, \ldots, \sigma_{2n}) \in S_{2n+1}$
$\sigma^{(-1)} \leftarrow (0, 1, \ldots, 2n)$
for $i$ from 0 to $n-1$
    $\sigma^{(i)} \leftarrow \sigma^{(i-1)}$
    $\sigma^{(i)}_{2i+1} \leftarrow [2i+1+s_i]_{2i+3}$, $\sigma^{(i)}_{2(i+1)} \leftarrow [2(i+1)+s_i]_{2i+3}$
    for $j$ from 0 to $2i$
        $\sigma^{(i)}_j \leftarrow [\sigma^{(i-1)}_j + s_i]_{2i+3}$
```

To prove that the mapping Π_3 is a DPM we use an analog of Lemma 4.5 to express the coordinates σ_i in the output σ nonrecursively in terms of the input symbols $s_0, \ldots, s_{n-1}$.

Lemma 5.3: Let $\Pi_3(\boldsymbol{s}) = \sigma$. Then for all $i = 0, \ldots, n-1$,

$$\begin{aligned}\sigma_0 &= s_0 + \cdots + s_{n-1},\\ \sigma_{2i+1} &= [2i+1+s_i]_{2i+3} + s_{i+1} + \cdots + s_{n-1},\\ \sigma_{2(i+1)} &= [2(i+1)+s_i]_{2i+3} + s_{i+1} + \cdots + s_{n-1}.\end{aligned}$$

Proof: The proof is similar to the proof of Lemma 4.5. We note that $[2i+1+s_i]_{2i+3} \leq 2i+2$ and hence

$$[[2i+1+s_i]_{2i+3} + s_{i+1}]_{2i+5} = [2i+1+s_i]_{2i+3} + s_{i+1},$$

that is the modulo operation need not be performed. Similar arguments hold for $\sigma_{2(i+1)}$, resulting in the nonrecursive formulae presented in the lemma. ■

Using this lemma, we can prove the following proposition.

Proposition 5.4: The mapping Π_3 from $\mathbb{Z}_3^n$ to S_{2n+1} is a DPM, that is, $d(\Pi_3(\boldsymbol{s}), \Pi_3(\boldsymbol{s}')) \geq d(\boldsymbol{s}, \boldsymbol{s}')$.

Proof: The idea of the proof is as follows. We first show that $s_i \neq s'_i$ implies that either at least one of $\sigma_{2i+t} - \sigma'_{2i+t}$, $t = 1, 2$ is nonzero, or if both are zero then $\sigma_{2i} \neq \sigma'_{2i}$. In the latter case, if we additionally have $s_{i-1} \neq s'_{i-1}$ then we show that $\sigma_{2i-1} \neq \sigma'_{2i-1}$.

For $t = 1, 2$ we can write,

$$\sigma_{2i+t} = [2i+t+s_i]_{2i+3} + \sum_{l=i+1}^{n-1} s_l$$

Similar equations hold for σ'_{2i+t}. Let $\Delta_i = \sum_{l=i}^{n-1} s_l - s'_l$. Thus, for $t = 1, 2$ we get,

$$\sigma_{2i} - \sigma'_{2i} = [2i + s_{i-1}]_{2i+1} - [2i + s'_{i-1}]_{2i+1} + \Delta_i, \tag{2}$$

$$\sigma_{2i+t} - \sigma'_{2i+t} = [2i+t+s_i]_{2i+3} - [2i+t+s'_i]_{2i+3} + \Delta_{i+1}, \tag{3}$$

Thus, if $s_i \neq s'_i$ and $\Delta_{i+1} = 0$ then we get $\sigma_{2i+t} \neq \sigma'_{2i+t}$ for both $t = 1, 2$. If $\Delta_{i+1} \neq 0$ and for at least one of $t = 1, 2$ we have $\sigma_{2i+t} \neq \sigma'_{2i+t}$ then we have proved the needed DPM condition. The only case that we need to consider now is the case where $\Delta_{i+1} = 0$, $\sigma_{2i+t} = \sigma'_{2i+t}$ for both $t = 1, 2$. A case by case analysis show that the only condition when this can happen is when $|s_i - s'_i| = 2$. In this case, we can write Δ_{i+1} in a compact form

$$\Delta_{i+1} = sgn(s_i - s'_i)(2i+1),$$

where $|s_i - s'_i| = 2$ and $sgn(x)$ is the sign function

$$sgn(x) = \begin{cases} 1, & x > 0,\\ 0, & x = 0,\\ -1, & x < 0.\end{cases}$$

Using (2) we get that $\sigma_{2i} \neq \sigma'_{2i}$. If in addition we have that $s_{i-1} \neq s'_{i-1}$, we show that we also have $\sigma_{2i-1} \neq \sigma'_{2i-1}$. We get

$$\begin{aligned}\sigma_{2i-1} - \sigma'_{2i-1} &= [2i-1+s_{i-1}]_{2i+1} - [2i-1+s'_{i-1}]_{2i+1} +\\ &\quad (s_i - s'_i) + \Delta_{i+1}\\ &= [2i-1+s_{i-1}]_{2i+1} - [2i-1+s'_{i-1}]_{2i+1} +\\ &\quad sgn(s_i - s'_i)(|s_i - s'_i| + 2i + 1).\end{aligned}$$

Now, the last term in the above equation is always $\pm(2i+3)$ while the difference of the first two terms is

$$[2i-1+s_{i-1}]_{2i+1} - [2i-1+s'_{i-1}]_{2i+1} \in \{\pm 1, \pm 2i, \pm(2i-1)\}.$$

This proves that $\sigma_{2i-1} \neq \sigma'_{2i-1}$. ■

A. Estimating the ternary symbols from the permutation vector

The estimation of the ternary symbols from the received vector is, not surprisingly, more computationally intensive than the corresponding one in Section IV-A.

Lemma 5.5: Let $\Pi_3(\boldsymbol{s}) = \sigma$. The differences between the symbols $\{\sigma_{2i+1}, \sigma_{2(i+1)}\}$ and $\{\sigma_{2j+1}, \sigma_{2(j+1)}\}$ for $j > i$ satisfies the following conditions. For $l \in \{2i+1, 2(i+1)\}$,

$$\sigma_l - \sigma_{2j+1} \begin{cases} < 0, & \text{if } s_j \in \{0, 1\},\\ > 0, & \text{if } s_j = 2,\end{cases}$$

$$\sigma_l - \sigma_{2(j+1)} \begin{cases} < 0, & \text{if } s_j = 0,\\ > 0, & \text{if } s_j \in \{1, 2\}.\end{cases}$$

Proof: We show the proof for only the case $\sigma_{2i+1} - \sigma_{2j+1}$ since the other cases are similar. Using Lemma 5.3 we get

$$\sigma_{2i+1} - \sigma_{2j+1} = [2i+1+s_i]_{2i+3} + \sum_{l=i+1}^{j-1} s_l + s_j - [2j+1+s_j]_{2j+3}.$$

For $s_j \in \{0,1\}$, we get $[2j+1+s_j]_{2j+3} = 2j+1+s_j$. Since $[2i+1+s_i]_{2i+3} + \sum_{l=i+1}^{j-1} s_l \leq 2i+2+2(j-1-i) = 2j$, we get that the RHS of the above equation is strictly negative. For $s_j = 2$, we get $[2j+1+s_j]_{2j+3} = 0$ and hence the RHS is always strictly positive. ■

This lemma suggests the following algorithm to determine the ternary symbol s_j. Let $\boldsymbol{y} = (y_0, \ldots, y_{2n})$ in $\{\mathbb{Z}_{2n+1} \cup \varepsilon\}^{2n+1}$ be the received vector. For $l < 2j+1$, if y_l, y_{2j+1}, y_{2j+2} are not erasures then we take the differences $y_l - y_{2j+1}$ and $y_l - y_{2(j+1)}$ and declare $s_j = 0$ if both the differences are negative, $s_j = 2$ if both the differences are positive, and $s_j = 1$ otherwise. We formalize this procedure in the following algorithm. This algorithm corresponds to Algorithm 4.7 of Section IV-A.

Algorithm *5.6:* Estimate ternary symbols from $\boldsymbol{y}$

```
**Input**: $\boldsymbol{y} = (y_0, \ldots, y_{2n}) \in \{\mathbb{Z}_{2n+1} \cup \varepsilon\}^{2n+1}$
**Output**: $\hat{\boldsymbol{s}} = (\hat{s}_0, \ldots, \hat{s}_{n-1}) \in \{\mathbb{Z}_3 \cup \varepsilon\}^n$
$L_0 \leftarrow \phi$, the empty set
**for** $j$ from 1 to $n$
    $L_j \leftarrow L_{j-1} \cup \{l : y_l \neq \varepsilon,\, l \in \{2(j-1), 2(j-1)-1\}\}$
    **if** $y_{2j} = \varepsilon$ or $y_{2j-1} = \varepsilon$ then $\hat{s}_{j-1} \leftarrow \varepsilon$
    **else**
        $t = (t_0, t_1, t_2) \leftarrow (0,0,0)$
        **for** each $l$ in $L_j$
            $p = (p_0, p_1) \leftarrow (y_l - y_{2j-1}, y_l - y_{2j})$
            **if** $p_0 < 0$ and $p_1 < 0$ then $t_0 \leftarrow t_0 + 1$
            **elseif** $p_0 < 0$ and $p_1 > 0$ then $t_1 \leftarrow t_1 + 1$
            **elseif** $p_0 > 0$ and $p_1 > 0$ then $t_2 \leftarrow t_2 + 1$
        **if** $t = (0,0,0)$ then $\hat{s}_{j-1} \leftarrow \varepsilon$
        **else** $\hat{s}_{j-1} \leftarrow \arg\max\{t_l : l \in \{0,1,2\}\}$
```

The maximum sum of the sizes of L_j is bounded as $\sum_{j=1}^{n} |L_j| \leq 1+3+\cdots+2n-1 = n^2$. Hence the maximum number of comparisons required is $8n+6n^2$, and the maximum number of subtractions and additions required is $3n^2$. Using at most a constant size of $|L_j| \leq \ell$ leads to less computations, at the loss of reliability of the symbols in the higher indices.

B. Estimating ternary symbols on an erasure channel

An analog of Lemma 4.8 allows us to adopt a simpler decoding procedure in an erasure channel. It relies on the following lemma.

Lemma 5.7: Let $\boldsymbol{s} \in \mathbb{Z}_3^n$, $\sigma = \Pi_3(\boldsymbol{s})$. If $\sigma_{2i+1} = 0$ or $\sigma_{2(i+1)} = 0$ then $s_i = 2$ or $s_i = 1$, resp., and $s_j = 0,\ j > i$. The proof of this lemma is very similar to the proof of Lemma 4.8.

Remark: If the demodulator can provide soft information on the reliability of the symbols, then Algorithms 4.7 and 5.6 can be simplified by fixing $|L_j| = 1$ and retaining only the most reliable symbol from the received symbols at step j. This also ensures correct estimation of the succeeding digits, if the corresponding symbols in the received vector are not in error or in erasure.

VI. Simulations

We consider the PLC channel with M-FSK modulation and with only background noise, for simplicity. The transmitted word is represented as an $M \times M$ $\{0,1\}$-matrix, where M is the length of the permutation. A 1 in the i-th row and j-th column indicates that the permutation symbol i is sent at time j. Since we are considering hard decision decoding, we simulate background noise by flipping the value of any entry of the matrix with a probability p. The codewords are chosen at random from BCH codes over the finite fields $\mathbb{F}_q$ for $q = 2,3,4$. For the maps Π_0, Π_1, the permutation symbol at time i is taken to be $i+1$ if the $(i+1,i)$-th entry of the received matrix is 1; it is assumed ε if all entries $(j,i), j \leq i$ are 0; otherwise it is assumed j if any (j,i)-th entry is 1 for $j \leq i$. For maps Π_2, Π_3 we set the permutation symbol at time i to ε if the column i does not contain exactly one 1. Fig. 1 shows the symbol error and erasure rate of the different estimation algorithms, after decoding the estimated symbols with a bounded distance error and erasure decoder.

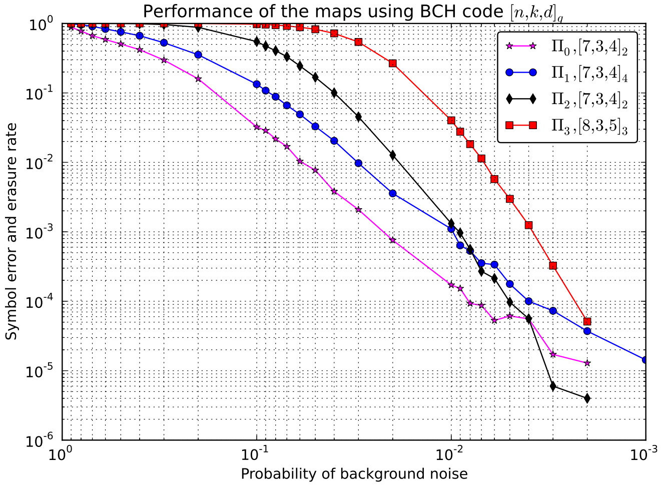

Fig. 1. Symbol error and erasure rates under background noise

VII. Discussion and Conclusion

We provided several different mappings from q-ary vectors in $\mathbb{Z}_q^n$ to permutations in S_N. The main focus of using such mappings was to implement simple estimation algorithms in the permutation space and provide the estimated digits to the q-ary code where efficient decoding algorithms can be implemented. Since the length $N = q + Q(n-1)$, $Q = \lceil \log_2 q \rceil$, the information rate of the codes decreases by a factor of approximately $1/\lceil \log_2 q \rceil$ for all the algorithms. We believe that it should be possible to generalize the map Π_3 from ternary vectors to the permutation space to a DPM $\Pi_q : \mathbb{Z}_q^n \to S_N$, by using an additional Q symbols at every iteration of the DPM. Hence we have the following conjecture.

Conjecture 7.1: Let $N = q + \lceil \log_2 q \rceil (n-1)$. Map the q-ary vector $(s_0, \ldots, s_{n-1})$ to $(\sigma_0, \ldots, \sigma_{N-1})$ by Π_q as described below:

```
$\sigma^{(0)} \leftarrow ([0+s_0]_q, \ldots, [q-1+s_0]_q, q, \ldots, q+Q(n-1)-1)$
**for** $i$ from 0 to $n-2$
    $\sigma^{(i+1)} \leftarrow \sigma^{(i)}$
    $\sigma_j^{(i+1)} \leftarrow [\sigma_j^{(i)} + s_{i+1}]_{Q(i+1)+q}, \; j \leq q+Q(i+1)-1.$
```

Then Π_q is a DPM.

This conjecture has been verified for lengths up to 6 and $q \leq 8$ via numerical calculations. The case $q = 8$ is the first case where $\log_2 8 = 3$ additional symbols are required for preserving the distance in the mapping. The following counterexample shows that 2 additional symbols for each 8-ary symbol do not suffice to make the mapping a DPM. The 8-ary vectors below have distance 5 while the mapped vectors have distance 4.

$$(0,0,4,5,0) \mapsto (9,10,11,12,13,0,1,2,5,6,7,8,3,4,14,15)$$
$$(3,7,2,0,7) \mapsto (9,10,11,12,13,0,1,2,14,15,7,8,3,4,5,6)$$

References

[1] R. F. Bailey, "Error-correcting codes from permutation groups", *Discrete Math.* vol. 309, pp. 4253–4265, 2009.

[2] I. F. Blake, G. Cohen and M. Deza, "Coding with permutations", *Inf. and Contr.*, vol. 43, pp. 1–19, 1979.

[3] J. C. Chang, "Distance-increasing mappings from binary vectors to permutations that increase hamming distances by at least two", *IEEE Trans. Inf. Theory*, vol. 52, pp. 1683–1689, April 2006.

[4] S. Huczynska, "Powerline communication and the 36 officers problem", *Phil. Trans. R. Soc. A*, vol. 364, pp. 3199–3214, 2006.

[5] K. Lee, "Distance-increasing maps of all lengths by simple mapping algorithms", *IEEE Trans. Inf. Theory*, vol. 52, pp. 3344–3348, July 2006.

[6] J. Lin, J. Chang, R. Chen, T. Kløve, "Distance-preserving and distance-increasing mappings from ternary vectors to permutations", *IEEE Trans. Inf. Theory*, vol. 54, pp. 1334–1339, March 2008.

[7] T. G. Swart and H. C. Ferreira, "Decoding distance-preserving permutation codes for power-line communications", *Africon 2007*, pp. 1–7.

[8] A. J. H. Vinck, "Coded modulation for power line communications", *AEU Int. J. of Elec. and Comm.,* vol. 54, pp. 45–49, January 2000.