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 Gaussian 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 $\{0, \ldots, N-1\}$. Each element $\sigma$ of $S_N$ is written as a vector $\sigma = (\sigma_0, \ldots, \sigma_{N-1})$, 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 $\mathbb{Z}_q^n = \{0, \ldots, q-1\}^n$ 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, $d(\Pi(x), \Pi(y)) \geq d(x, y)$ for any vectors $x, y \in \mathbb{Z}_q^n$. A distance-increasing map $\Pi : \mathbb{Z}_q^n \to S_N$ strictly increases the Hamming distance, that is, $d(\Pi(x), \Pi(y)) > d(x, y)$ for any distinct vectors $x, y \in \mathbb{Z}_q^n$.

An upper bound on the size of a permutation code with minimum distance $d$ is given by $N!/d!$. 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 $1/\left\lceil \log_2 q \right\rceil$, 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 $b = (b_0, \ldots, b_{n-1})$ of length $n$ to a vector $\sigma = (\sigma_0, \ldots, \sigma_n)$ 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 \( \Pi_0 \) denote this mapping. The example below illustrates the map of \( b = (1, 1, 0, 1) \) to the permutation vector \( (1, 2, 0, 4, 3) \). For brevity of exposition we write the vector \( \sigma \) in a compact form, \( \sigma = 12043 \). Underlined portions denote the affected symbols.

**Example 2.1:** \( 01234 \overset{b_0=1}{\rightarrow} 10234 \overset{b_1=1}{\rightarrow} 12034 \overset{b_2=0}{\rightarrow} 12034 \).

**Algorithm 2.2:** DIM \( \Pi_0 \) from \( \mathbb{Z}_2^n \) to \( S_{n+1} \)

**Input:** \( b = (b_0, \ldots, b_{n-1}) \in \mathbb{Z}_2^n \)

**Output:** \( \sigma = (\sigma_0, \ldots, \sigma_n) \in S_{n+1} \)

**Algorithm 2.4:** Estimating bits from \( y \)

**Input:** \( y = (y_0, \ldots, y_n) \in \{\mathbb{Z}_{n+1} \cup \varepsilon\}^{n+1} \)

**Output:** \( b = (b_0, \ldots, b_{n-1}) \in \{\mathbb{Z}_2 \cup \varepsilon\}^n \)

**Algorithm 3.2:** DIM \( \Pi_1 \) from \( \mathbb{Z}_{2m}^n \) to \( S_{mn+1} \)

**Input:** \( s = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_{2m}^n \)

**Output:** \( \sigma = (\sigma_0, \ldots, \sigma_{mn}) \in S_{mn+1} \)

**Algorithm 3.2:** DIM \( \Pi_1 \) from \( \mathbb{Z}_{2m}^n \) to \( S_{mn+1} \)

**Input:** \( s = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_{2m}^n \)

**Output:** \( \sigma = (\sigma_0, \ldots, \sigma_{mn}) \in S_{mn+1} \)

**for** \( i = 0 \) to \( n - 1 \)

**if** \( y_i = i + 1 \) then \( b_i \leftarrow 1 \)

**else** \( y_i < i + 1 \) then \( b_i \leftarrow 0 \)

**else** \( b_i \leftarrow \varepsilon \).

The estimate \( \hat{b} \) 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 \( n_e \) satisfy the condition \( 2n_e + n_e < d \), 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 \( b_i \).

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 \( \tilde{b} \) of length \( n - 3 \), but in reverse order of the DIM. If \( \hat{\sigma}^{(-1)} = (3, \ldots, n) \) then \( b_0 \) flips \( \hat{\sigma}_{n-3}^{-1} \) and \( \hat{\sigma}_{n-4}^{-1} \), \( b_1 \) flips \( \hat{\sigma}_{n-5}^{-1} \), and so on.

**III. DIM FROM \( 2^m \)-ARY VECTORS TO PERMUTATIONS**

In this section we develop 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 \( \Pi_1 \). 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 \( 0 \rightarrow 00, 1 \rightarrow 01, 2 \rightarrow 10, \) and \( 3 \rightarrow 11 \). The vector \( s = 132 \) is mapped to the permutation vector 0234516 in the following sequence of steps: 0123456 \( \overset{b_0}{\rightarrow} 01 \overset{b_1}{\rightarrow} 023456 \overset{b_2}{\rightarrow} 132 \overset{b_3}{\rightarrow} 123456 \overset{b_4}{\rightarrow} 123456 \). The underlined portions denote the affected symbols.

**Algorithm 3.2:** DIM \( \Pi_1 \) from \( \mathbb{Z}_{2m}^n \) to \( S_{mn+1} \)

**Input:** \( s = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_{2m}^n \)

**Output:** \( \sigma = (\sigma_0, \ldots, \sigma_{mn}) \in S_{mn+1} \)

**for** \( i = 0 \) to \( n - 1 \)

**for** \( j = 0 \) to \( m - 1 \)

**if** \( b_i = 0 \) then \( \sigma_{im+j} \leftarrow \sigma_{im+j}^{-1} \)

**else** \( \sigma_{im+j} \leftarrow \sigma_{im+j}^{-1} \)

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 \(\sigma = (\sigma_0 + i, \ldots, \sigma_{i+1}, l+1+i, \ldots, l+j+i) \mod (l+j+1)\) 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 \(\Sigma\) results in a vector \(\Sigma + i\) which is a cyclic shift of \(\Sigma\) to the left by \(i\) positions. Hence \(\Sigma + i\) is also in \(S_{l+j+1}\). Considered as an unordered tuple, the elements of \(\sigma\) are the same as the elements of \(\Sigma + i\) and hence \(\sigma\) 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 \(b = (b_0, \ldots, b_{n-1})\) 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 \(\sigma^{(-1)}\) modulo \(i+2\), where \(\sigma^{(-1)}\) is the vector resulting from the previous iteration. Denote the DPM by \(\Pi_i\). The example below illustrates the procedure.

**Example 4.2:** We map \(1101\) to \(32140\) as follows:

```
01234 \rightarrow 10234 \rightarrow 21034 \rightarrow 21034 \rightarrow 32140.
```

**Algorithm 4.3:** DPM \(\Pi_2\) from \(\mathbb{Z}_2^n\) to \(S_{n+1}\)

**Input:** \(b = (b_0, \ldots, b_{n-1}) \in \mathbb{Z}_2^n\)

**Output:** \(\sigma = (\sigma_0, \ldots, \sigma_n) \in S_{n+1}\)

\(\sigma^{(-1)} \leftarrow (0, 1, \ldots, n)\)

for \(i\) from 0 to \(n-1\)

\(\sigma^{(i)} \leftarrow \sigma^{(-1)} + b_i \mod (i+2)\)

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

Before providing the proof of the proposition we determine the output of the mapping \(\Pi_2\) 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 \(a \mod p\). Recall that the binary vector \(b\) is considered as a \(\{0,1\}\)-vector over \(\mathbb{R}\).

**Lemma 4.5:** If \(\Pi_2(b) = \sigma = (\sigma_0, \ldots, \sigma_l)\), then

\(\sigma_0 = b_0 + \cdots + b_{n-1}\), \(\sigma_l = (l + b_{l-1}|_{l+1} + b_l + \cdots + b_{n-1}, \ l + 1, \ldots, n)\).

**Proof:** The output of the mapping \(\Pi_2\) is given by

\(\sigma_0 = \sum_{i=0}^{n-1} b_i\), \(\sigma_l = [l + b_{l-1}|_{l+1} + b_l + \cdots + b_{n-1}]_{n+1}, \ l > 0\).

For any \(l = 1, \ldots, n\), we have that \([l + b_{l-1}|_{l+1} + b_l + \cdots + b_{n-1}]_{n+1}\) is a permutation of \((0, 1, \ldots, n)\), and the modulo operation is performed on each coordinate of the vector.

**Proof of Proposition 4.4:** Let \(b = (b_0, \ldots, b_{n-1})\) and \(b' = (b_0', \ldots, b_{n-1}')\), be \(\{0,1\}\)-vectors over \(\mathbb{R}\). Suppose \(b_{l-1} \neq b'_{l-1}\). Then we show that either \(\sigma_i \neq \sigma'_i\) or \(\sigma_{i-1} \neq \sigma'_{i-1}\).

Let \(\Delta_i = \sum_{j=0}^{n-1} b_{i} - b'_{i}\). 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 \(i = -\Delta_{i-1}\). 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 \(i = -\Delta_{i-1}\). 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 all \(k\) 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 \(i = 1, \ldots, n\), we write \(\sigma_l - \sigma'_l = (b_{l-1} - b'_{l-1}) l + \Delta_i\). Using \(\sigma_{i+k} - \sigma'_{i+k} = (b_{i+k-1} - b'_{i+k-1}) (i + k) + \Delta_{i+k} = 0\), we get \(\sigma_l - \sigma'_l = (b_{i-k-1} - b'_{i-k-1}) i + \Delta_i - \Delta_{i+k} + (b_{i+k-1} - b'_{i+k-1}) (i + k + 1)\).

Since \(|b_{i-1} - b'_{i-1}| i + \Delta_i - \Delta_{i+k-1} i \leq i + k - 1\) and the last term of \(\sigma_l - \sigma'_l\) is \(\pm(i + k + 1)\), we get \(\sigma_l \neq \sigma'_l\).

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 \(\Pi_2(b) = \sigma\). The difference between any two coordinates \(\sigma_i, \sigma_j\) for \(i > j\) 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,

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

The statements in the lemma follow from the observation that \(|i + b_{i-1}|_{i+1} + \sum_{l=1}^{j-2} b_l \leq j - 1\).

Let the received vector from the channel be denoted by \(y \in \mathbb{Z}_{n+1} \cup \varepsilon^n\). 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 \(b_{j-1} = \varepsilon\). Algorithm 4.7 below describes the procedure.

The term \(t > 0\) in the algorithm corresponds to performing a majority vote for the estimate \(b_j\) for each \(j = 0, \ldots, n - 1\). Algorithm 4.7 requires at most \(n(n + 1)/2\) additions and subtractions, and \(3n + \frac{3}{2}(n + 1)\) comparisons. By restricting \(|L_j|\)
to at most a constant number, say $\ell$, 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 $\hat{b}$ from $y$**

**Input:** $y = (y_0, \ldots, y_n) \in \{Z_{n+1} \cup \varepsilon\}^{n+1}$

**Output:** $\hat{b} = (b_0, b_1, \ldots, b_{n-1}) \in \{Z_2 \cup \varepsilon\}^n$

$L_0 \leftarrow \emptyset$, 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 $b_{j-1} \leftarrow \varepsilon$

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 $b_{j-1} \leftarrow 1$

elseif $t < 0$ then $b_{j-1} \leftarrow 0$

else $b_{j-1} \leftarrow \varepsilon$

**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 $\hat{b}$ is at most the number of erasures in the received symbols $y$. In addition, if the symbol 0 of $S_{n+1}$ is present in the received vector $y$, 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 $b \in \mathbb{Z}_2^n$ and let $\sigma = \Pi_2(b)$. 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 $s = (s_0, \ldots, s_{n-1})$, 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 $\Pi_3$ 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 $\Pi_3$ as follows. 0123456 $s_0 \leftarrow 1$ 1203456 $s_1 \leftarrow 2$ 3420156 $s_2 \leftarrow 1$ 4531260.

**Algorithm 5.2:** DPM $\Pi_3$ from $\mathbb{Z}_3^n$ to $S_{2n+1}$

**Input:** $s = (s_0, \ldots, s_{n-1}) \in \mathbb{Z}_3^n$

**Output:** $\sigma = (\sigma_0, \ldots, \sigma_{2n}) \in S_{2n+1}$

\[
\begin{align*}
\sigma^{(-1)} & \leftarrow (0, 1, \ldots, 2n) \\
\text{for } i \text{ from } 0 \text{ to } n - 1: & \\
\sigma^{(i)} & \leftarrow \sigma^{(i-1)} \\
\sigma^{(i)}_{2i+1} & \leftarrow [2i+1 + s_i]_{2i+3}, \quad \sigma^{(i)}_{2(i+1)} & \leftarrow [2(i+1) + s_i]_{2i+3} \\
\text{for } j \text{ from } 0 \text{ to } 2i: & \\
\sigma^{(i)}_j & \leftarrow [\sigma^{(i-1)}_{j+1} + s_i]_{2i+3} \\
\end{align*}
\]

To prove that the mapping $\Pi_3$ is a DPM we use an analog of Lemma 4.5 to express the coordinates $\sigma_i$ in the output $\sigma$ nonrecursively in terms of the input symbols $s_0, \ldots, s_{n-1}$.

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

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

Using this lemma, we can prove the following proposition.

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

**Idea of Proof:** We first show that $s_i \neq s_i'$ implies that either at least one of $\sigma_{2i+j} - \sigma_{2i+j'}$, $j = 1, 2$ 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 $\sigma_{2i-1} \neq \sigma'_{2i-1}$.

**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 $\Pi_3(s) = \sigma$. The differences between the symbols $\{\sigma_{2i+1}, \sigma_{2(i+1)}\}$ and $\{\sigma_{2i-1}, \sigma_{2(i+1)}\}$ for $j > i$ satisfies the following conditions. For $i \in \{2i + 1, 2(i + 1)\}$,

\[
\begin{align*}
\sigma_{2i+1} - \sigma_{2j+1} & \leftarrow \begin{cases} 
< 0, & \text{if } s_j \in \{0, 1\}, \\
> 0, & \text{if } s_j = 2;
\end{cases} \\
\sigma_{2i+1} - \sigma_{2j+1} & \leftarrow \begin{cases} 
< 0, & \text{if } s_j = 0, \\
> 0, & \text{if } s_j \in \{1, 2\}.
\end{cases}
\end{align*}
\]

**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 $y = (y_0, \ldots, y_{2n})$ in $\{Z_{2n+1} \cup \varepsilon\}^{2n+1}$ 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 $y_l - y_{2j+2}$ 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.
The symbol error and erasure rate $\epsilon$ of the matrix with a probability $\epsilon$ of the map can be simplified by fixing $\Pi_3$ from ternary vectors to the permutation space to a DPM $\Pi_\chi$ : $\mathbb{Z}_q^n \rightarrow S_N$, by using an additional $Q$ symbols at every iteration of the DPM. Hence we have the following: Conjecture: Map $\sigma$ to $\sigma_{\Pi_3}$ by $\Pi_\chi$: $\sigma^{(t)} \leftarrow ([0+1] \sigma[q], \ldots, [q-1+1] \sigma[q], \ldots, q+Q(n-1)$ for $t$ from 0 to $n-2$ $\sigma^{(t+1)} \leftarrow \sigma^{(t)} + s(t)^+ + s(t)^-$, $j \leq q + (i+1) - 1$. Then $\Pi_\chi$ is a DPM.

**REFERENCES**


