An Efficient Decoding Algorithm of Matrix Partition Codes

Feng Chen, Samuel Cheng, *Member, IEEE*

***Abstract*—Recently, a multiple Slepian-Wolf Coding method called Matrix Partition Codes has been introduced. However, no explicit decoding algorithm has been established so far. In this letter, we propose a two-step decoding algorithm which just needs an extra one-time preprocessing ($O(rsn|\mathcal{D}|+r|\mathcal{D}|\log(|\mathcal{D}|)+n^2sm)$). To illustrate the efficiency of our algorithm, we analyze the time complexity of our algorithm, and the results show that the complexity of our method ($O(r\log(|\mathcal{D}|)+msn)$) is exponentially better than the brute-force search method ($O(|\mathbb{F}|^n|\mathcal{D}|msn)$).**

***Index Terms*—multiple Slepian-Wolf Coding, Matrix Partition Codes, Confined-Correlated, decoding.**

I. Introduction

DISTRIBUTED Source Coding (DSC) refers to separate compression and joint decompression of multiple correlated sources, and it can be illustrated as Figure 1. DSC started as an information-theoretical problem in the renowned 1973 paper of Slepian and Wolf [1], and their theory is called Slepian-Wolf (SW) coding. SW coding is a lossless data compression technique, which encodes each source separately and decodes jointly. By this, the computation burdens are shifted from the encoded sides to the decoded side. A large number of SW coding schemes have been proposed [2], [3].

Syndrome based SW coding is a highly efficient linear method, and it has been widely used. Wyner is the first who realized this [4], and took computed syndromes as compressed sources. Since then, many researchers have been focusing on this interesting topic [2], [5]. Despite this great progress, most of them are only concentrating on two sources except for a few exceptions [6], [7].

In [8], Ma and Cheng introduced a truly lossless multiple SW coding based on syndromes called Hamming Codes for Multiple Sources (HCMSs), which is a perfect code solution for Hamming sources. Then in [9], they extended their input sources from Hamming sources to arbitrary fields, called Confined-Correlated Sources with Deviation Symmetry (CCSDS), and introduced Matrix Partition Codes to handle these sources. Despite this progress, they only showed the optimality of their codes and did not offer any efficient decoding algorithm. In this paper, we start from the construction of Partition Matrix Codes to derive a precise decoding algorithm. To illustrate the efficiency of our algorithm, we will analyze its running time complexity.

This paper is organized as follows. In section II, we will review the Confined-Correlated Sources with Deviation Symmetry, and Matrix Partition Codes. In section III, we will introduce our method, and analyze it in section IV. We conclude our work in section V.

The authors are with the School of Electrical and Computer Engineering, University of Oklahoma, Tulsa, OK, 74135 USA (email: {achenfengb, samuel.cheng}@ou.edu).

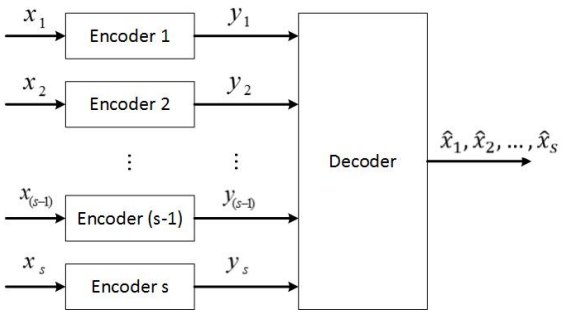

Fig. 1. Framework of Multiple Distributed Source Coding: each source $\mathbf{x_i}$ is encoded separately, and decode them jointly to get $\hat{\mathbf{x}}_\mathbf{i}$ in the common decoder.

II. Preliminary

Suppose we have a joint source $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_s)$ sampled uniformly from a set $\mathcal{S}$. To compress these sources based on syndrome coding, we will construct s parity-check matrices $(\mathbf{H}_1, \ldots, \mathbf{H}_s)$ to get the syndromes $(\mathbf{y}_1, \ldots, \mathbf{y}_s)$, i.e.,

$$\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_s) = (\mathbf{H}_1\mathbf{x}_1, \ldots, \mathbf{H}_s\mathbf{x}_s), \tag{1}$$

where $\mathbf{x}_i \in \mathbb{F}^n$, $\mathbf{H}_i \in \mathbb{F}^{m_i \times n}$, and $\mathbf{y}_i \in \mathbb{F}^{m_i}$. As mentioned in section I, a Partition Matrix Code is truly lossless, hence the function defined by (1) must be injective in domain $\mathcal{S}$. In the rest of this section, we will briefly introduce the input source space and parity-check matrices.

If there exists $\mathcal{D} \subseteq \mathbb{F}^n \times \cdots \times \mathbb{F}^n$, such that

$$\mathcal{S} = \{(\mathbf{v}, \ldots, \mathbf{v}) + \delta | \mathbf{v} \in \mathbb{F}^\mathbf{n}, \delta \in \mathcal{D}\}, \tag{2}$$

then we call $\mathcal{S}$ as the Confined-Correlated Sources with Deviation Symmetry (CCSSD), and denote it as $\mathcal{S}(\mathcal{D})$.

Generally, $\mathcal{D}$ can be an arbitrary subset of $\mathbb{F}^n \times \cdots \times \mathbb{F}^n$, however, according to (2), δ and $\delta + (\mathbf{v}, \ldots, \mathbf{v})$ will result in the same source tuples in $\mathcal{S}(\mathcal{D})$, $\forall \mathbf{v} \in \mathbb{F}^n$. To reduce this redundancy, the following constraint is imposed on $\mathcal{D}$,

$$\delta \in \mathcal{D} \Rightarrow \delta + (\mathbf{v}, \ldots, \mathbf{v}) \notin \mathcal{D}, \forall \text{ non-zero } \mathbf{v} \in \mathbb{F}^\mathbf{n}. \tag{3}$$

The above constraint guarantees that $\mathcal{S}(\mathcal{D})$ is generated by the least size $\mathcal{D}$, and

$$|\mathcal{S}| = |\mathbb{F}|^n|\mathcal{D}|. \tag{4}$$

Example 1 (Hamming Sources)**.** A Hamming Source $\mathcal{S}$ over $\mathbb{F}$ defined by [8]

$$\begin{aligned}\mathcal{S} = \{(\mathbf{v},\dots,\mathbf{v}) + (\underbrace{\mathbf{0},\dots,a\mathbf{e}_j}_{i \text{ items}},\dots,\mathbf{0}) | a \in \mathbb{F},\\ 1 \le i \le s, 1 \le j \le n\},\end{aligned} \tag{5}$$

which is clearly a CCSDS, where $\mathbf{e}_j$ is a length-n vector with all zeros except the j^{th} element being 1. For $s \ge 3$, $\mathcal{D}$ can be simply chosen as

$$\mathcal{D} = \{(\underbrace{\mathbf{0},\dots,a\mathbf{e}_j}_{\text{i items}},\dots,\mathbf{0}) | a \in \mathbb{F}, 1 \le i \le s, 1 \le j \le n\}, \tag{6}$$

and have

$$|\mathcal{S}| = |\mathbb{F}|^n (1 + s(|\mathbb{F}| - 1)n) \text{ for finite } \mathbb{F}. \tag{7}$$

For the parity-check matrices, we can construct them by the following theorem [9].

Theorem 1. *Let* $\mathbf{P}$ *be an* $r \times sn$ *matrix (*$r \in \mathbb{Z}_+$*) over* $\mathbb{F}$ *s.t.*

$$\mathbf{P}|_{\tilde{\mathcal{D}}} \textit{ is one to one}, \tag{8}$$

where

$$\tilde{\mathcal{D}} = \{(\mathbf{d}_1^{\mathbf{T}} \dots \mathbf{d}_s^{\mathbf{T}})^{\mathbf{T}} | (\mathbf{d}_1,\dots,\mathbf{d}_s) \in \mathcal{D}\}. \tag{9}$$

Suppose $\mathbf{P}$ *can be partitioned into*

$$\mathbf{P} = [\mathbf{Q}_1 | \dots | \mathbf{Q}_s] \textit{ s.t. } \mathbf{Q_1} + \dots + \mathbf{Q}_s = 0, \tag{10}$$

where all $\mathbf{Q}_i$ *are* $r \times n$ *matrices. Then, for any matrix* $\mathbf{T}$ *that*

$$\begin{pmatrix}\mathbf{Q}_1\\ \vdots\\ \mathbf{Q}_s\\ \mathbf{T}\end{pmatrix} \tag{11}$$

forms an injective matrix, we let $\mathbf{G}_i (1 \le i \le s)$ *be a row partition of* $\mathbf{T}$*, i.e.*

$$\mathbf{T} = \begin{pmatrix}\mathbf{G}_1\\ \vdots\\ \mathbf{G}_s\end{pmatrix}. \tag{12}$$

Parity Check (encoding) matrices $(\mathbf{H}_1,\dots,\mathbf{H}_s)$ *with*

$$null(\mathbf{H}_i) = null\begin{pmatrix}\mathbf{G}_i\\ \mathbf{Q}_i\end{pmatrix} \tag{13}$$

form a compression that name Matrix Partition Codes.

Remark 1. One may take $\mathbf{H}_i$ as a row basis matrix of $\begin{pmatrix}\mathbf{G}_i\\ \mathbf{Q}_i\end{pmatrix}$ to increase compression, i.e.

$$\mathbf{H}_i = \mathbf{U}_i \begin{pmatrix}\mathbf{G}_i\\ \mathbf{Q}_i\end{pmatrix}, \tag{14}$$

where $\mathbf{U}_i$ is an arbitrary invertible matrices with appropriate sizes for all i.

III. Proposed Method

For a Matrix Partition Code $(\mathbf{H}_1,\dots,\mathbf{H}_s)$ defined by Theorem 1 and the source space defined by (2), we will derive how to use $\mathbf{Y}$ to estimate $\mathbf{X}$ explicitly.

For simplicity, we will derive our method for the special case when $\mathbf{H}_i = \begin{pmatrix}\mathbf{G}_i\\ \mathbf{Q}_i\end{pmatrix}$, and will point out how it works for the general case (cf. Remark 1) by the end of this section.

Our algorithm can be divided into two steps: the first step is to find $\hat{\delta}$ in (2), the second step is to solve $\hat{\mathbf{v}}$, and the final step is to combine them to get $\hat{\mathbf{X}}$. We summarize the overall steps in Algorithm 1. In the rest of this section, we will explain our algorithm in detail.

Suppose $\mathbf{x}_i = \mathbf{v} + \mathbf{d}_i$, where $i = 1, \cdots, s$. According to (1), we have

$$\begin{aligned}\mathbf{y} &= \mathbf{y}_1 + \dots + \mathbf{y}_s\\ &= \mathbf{H}_1\mathbf{x}_1 + \dots + \mathbf{H}_s\mathbf{x}_s\\ &= \begin{pmatrix}\mathbf{G}_1\\ \mathbf{Q}_1\end{pmatrix}\mathbf{x}_1 + \dots + \begin{pmatrix}\mathbf{G}_s\\ \mathbf{Q}_s\end{pmatrix}\mathbf{x}_s.\end{aligned} \tag{15}$$

Let $\mathbf{y}'_i$ denotes the column vector containing the last r coefficients of $\mathbf{y}_i$, by (15), we have

$$\begin{aligned}\mathbf{y}' &= \mathbf{y}'_1 + \dots + \mathbf{y}'_s\\ &= \mathbf{Q}_1\mathbf{x}_1 + \dots + \mathbf{Q}_s\mathbf{x}_s\\ &= (\mathbf{Q}_1,\dots,\mathbf{Q}_s)\begin{pmatrix}\mathbf{v}+\mathbf{d_1}\\ \vdots\\ \mathbf{v}+\mathbf{d_s}\end{pmatrix}\\ &\overset{a}{=} (\mathbf{Q}_1 + \dots + \mathbf{Q}_s)\mathbf{v} + \mathbf{P}\begin{pmatrix}\mathbf{d}_1\\ \vdots\\ \mathbf{d}_s\end{pmatrix}\\ &\overset{b}{=} \mathbf{P}\begin{pmatrix}\mathbf{d}_1\\ \vdots\\ \mathbf{d}_s\end{pmatrix} = \mathbf{P}\tilde{\delta},\end{aligned} \tag{16}$$

where

$$\tilde{\delta} = \begin{pmatrix}\mathbf{d}_1\\ \vdots\\ \mathbf{d}_s\end{pmatrix}, \tag{17}$$

and steps (a) and (b) follow from (10). In (16), $\mathbf{d}_1,\dots,\mathbf{d}_s$ are unknown variables. Since $\mathbf{P}|_{\tilde{\mathcal{D}}}$ is invertible (generally $\mathbf{P}|_{\mathbb{F}^{sn}}$ is not invertible), it seems we could solve linear (16) directly. However, because the structure of $\mathcal{D}$ is unknown, so it is generally difficult to solve (16) directly. To remedy this, we establish a solution table: the table maps all the elements $\tilde{\delta} \in \tilde{\mathcal{D}}$ to their $\mathbf{y}'$ according to (16); moreover, we use the divide-and-conquer algorithm [10, chp. 4] to sort this table, and the order is determined by the value of $\mathbf{y}'_i$ $(i = 1,\dots,|\tilde{\mathcal{D}}|)$. Because $\mathbf{y}'_i$ is a vector, we first sort $\mathbf{y}'_i$ by their first elements (scalars), and if they are equal, then sort by their second elements, and so on so forth.

Once we compute $\mathbf{y}'$, we can solve (16) by the sorted table. We will use the binary search algorithm [10, chp. 2]

Algorithm 1 Decoding algorithm for Matrix Partition Codes: recover $(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_s)$ by syndromes $(\mathbf{y}_1, \ldots, \mathbf{y}_s)$.

Construct solution table according to (16) :

- Generate the set $\{(\tilde{\delta}_k, \mathbf{y}'_k) | \mathbf{y}'_k = \mathbf{P}\tilde{\delta}_k, \forall\ \tilde{\delta}_k \in \tilde{\mathcal{D}}\}$.
- Use the divide-and-conquer algorithm to sort the elements of above set according to the value of $\mathbf{y}'_k$, and establish the solution table.

Inputs: $\mathbf{y}_1, \ldots, \mathbf{y}_s$.

Solve (16) :

- Compute the summation: $\mathbf{y}' = \sum_{i=1}^{s} \mathbf{y}_i$.
- Use the binary search algorithm to look up $\mathbf{y}'$ in the solution table, and obtain its corresponding value of $\tilde{\delta}$, and $\hat{\mathbf{d}}_i (i = 1, \ldots, s)$.

Find the solution for $\mathbf{v}$:

- Substitute $\hat{\mathbf{y}}_i$ and $\hat{\mathbf{d}}_i (i = 1, \ldots, s)$ into (19).
- Solve $\hat{\mathbf{v}}$ by (20).

Compute the input source : $\hat{\mathbf{x}}_i = \hat{\mathbf{v}} + \hat{\mathbf{d}}_i$.

Output: $(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_s)$.

for searching, and because $\mathbf{P}|_{\tilde{\mathcal{D}}}$ is a one to one mapping, so for each $\mathbf{y}'$, we can find a unique record in this sorted table, and then obtain the solution $\tilde{\delta}$, hence we obtain $(\hat{\mathbf{d}}_1, \ldots, \hat{\mathbf{d}}_s)$.

After $(\hat{\mathbf{d}}_1, \ldots, \hat{\mathbf{d}}_s)$ is known, we can compute $\hat{\mathbf{v}}$. First, let's rewrite (1) in a vectorized form

$$\begin{pmatrix} \mathbf{y_1} \\ \vdots \\ \mathbf{y_s} \end{pmatrix} = \begin{pmatrix} \mathbf{H_1 x_1} \\ \vdots \\ \mathbf{H_s x_s} \end{pmatrix} = \begin{pmatrix} \mathbf{H_1}(\mathbf{v} + \hat{\mathbf{d}_1}) \\ \vdots \\ \mathbf{H_s}(\mathbf{v} + \hat{\mathbf{d}_s}) \end{pmatrix}. \tag{18}$$

After simple transforms, we have

$$\begin{pmatrix} \mathbf{y_1} - \mathbf{H_1}\hat{\mathbf{d}_1} \\ \vdots \\ \mathbf{y_s} - \mathbf{H_s}\hat{\mathbf{d}_s} \end{pmatrix} = \begin{pmatrix} \mathbf{H_1} \\ \vdots \\ \mathbf{H_s} \end{pmatrix} \mathbf{v} = \mathbf{H}\mathbf{v}. \tag{19}$$

From Theorem 1, we know $\mathbf{H}$ is an injective matrix, so it has a unique solution. We can solve (18) by

$$\hat{\mathbf{v}} = (\mathbf{H^T H})^{-1} \mathbf{H^T} \begin{pmatrix} \mathbf{y_1} - \mathbf{H_1}\hat{\mathbf{d}_1} \\ \vdots \\ \mathbf{y_s} - \mathbf{H_s}\hat{\mathbf{d}_s} \end{pmatrix}. \tag{20}$$

Finally, we obtain the codes

$$(\hat{\mathbf{x_1}}, \ldots, \hat{\mathbf{x_s}}) = \hat{\mathbf{v}} + (\hat{\mathbf{d}_1}, \ldots, \hat{\mathbf{d}_s}). \tag{21}$$

Remark 2. If $\mathbf{H}_i$ takes the general form as (14), then we need first left multiply $\mathbf{y}_i$ by $\mathbf{U}_i^{-1}$, i.e. $\tilde{\mathbf{y}}_i = \mathbf{U}_i^{-1}\mathbf{y}_i$ and use $\tilde{\mathbf{y}}_i$ instead of $\mathbf{y}_i$ in the following steps.

We will illustrate our method explicitly in this next example.

Example 2. (Generalized Hamming Code): $\mathbb{F} = \mathbb{Z}_5, a = 1, n = 6, s = 4$:

$$\begin{aligned}
\mathbf{H}_1 = \mathbf{Q}_1 &= \begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 1 & 2 & 3 & 4 \end{pmatrix}, \\
\mathbf{H}_2 = \mathbf{Q}_2 &= \begin{pmatrix} 4 & 0 & 2 & 4 & 2 & 4 \\ 0 & 2 & 2 & 3 & 1 & 1 \end{pmatrix}, \\
\mathbf{H}_3 = \mathbf{Q}_3 &= \begin{pmatrix} 2 & 0 & 3 & 1 & 3 & 1 \\ 0 & 4 & 3 & 3 & 2 & 4 \end{pmatrix}, \\
\mathbf{H}_4 = \mathbf{Q}_4 &= \begin{pmatrix} 3 & 0 & 4 & 4 & 3 & 3 \\ 0 & 3 & 4 & 2 & 4 & 1 \end{pmatrix}.
\end{aligned} \tag{22}$$

The matrix $\mathbf{P} = [\mathbf{Q_1}|\mathbf{Q_2}|\mathbf{Q_3}|\mathbf{Q_4}]$ consists of all nonzero vectors of $\mathbb{F}^2$ without repetition.

If the codeword is

$$(\mathbf{y}_1 \quad \mathbf{y}_2 \quad \mathbf{y}_3 \quad \mathbf{y}_4) = \left(\begin{pmatrix} 0 \\ 2 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right), \tag{23}$$

we have

$$\mathbf{y}' = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4 = \begin{pmatrix} 1 \\ 3 \end{pmatrix}. \tag{24}$$

By searching the solution table, we find the solution δ, i.e. $\hat{\mathbf{d}}_1, \hat{\mathbf{d}}_2, \hat{\mathbf{d}}_3, \hat{\mathbf{d}}_4$, that is equal to

$$\hat{\mathbf{d}_1}, \hat{\mathbf{d}_2}, \hat{\mathbf{d}_3} = (0,0,0,0,0,0)^{\mathbf{T}}, \hat{\mathbf{d}_3} = (0,0,0,1,0,0)^{\mathbf{T}}. \tag{25}$$

Then according to (19), we can obtain the equations

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 1 & 2 & 3 & 4 \\ 4 & 0 & 2 & 4 & 2 & 4 \\ 0 & 2 & 2 & 3 & 1 & 1 \\ 2 & 0 & 3 & 1 & 3 & 1 \\ 0 & 4 & 3 & 3 & 2 & 4 \\ 3 & 0 & 4 & 4 & 3 & 3 \\ 0 & 3 & 4 & 2 & 4 & 1 \end{pmatrix} \mathbf{v} = \begin{pmatrix} 0 \\ 2 \\ 3 \\ 4 \\ 3-1 \\ 2-3 \\ 0 \\ 0 \end{pmatrix}, \tag{26}$$

and then compute

$$\mathbf{v} = (0 \quad 3 \quad 2 \quad 3 \quad 4 \quad 1)^{\mathbf{T}} \tag{27}$$

by (20).

Finally, we have

$$(\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4) = \left(\begin{pmatrix} 0 \\ 3 \\ 2 \\ 3 \\ 4 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 3 \\ 2 \\ 3 \\ 4 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 3 \\ 2 \\ 4 \\ 4 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 3 \\ 2 \\ 3 \\ 4 \\ 1 \end{pmatrix} \right) \tag{28}$$

from the (21).

Note that above computations are implemented in field $\mathbb{Z}_5$.

In the following section, we will analyze the complexity of our algorithm.

IV. Complexity Analysis of Our Method

For the simplicity of analysis, we assume that

1) Scalar addition, multiplication, and comparison consume a same unit time called an operation.
2) Each $\mathbf{H}_i$ in (14) has the same size $m \times n$, i.e. $\mathbf{H}_i \in \mathbb{F}^{m\times n}$.
3) The process of computation is ordinary, which means no tricky algorithm has been taken. Take n square matrix mutiplication for instance, its complexity is $O(n^3)$ for ordinay computation, but according to [11], its complexity can be bounded by $O(n^{2.3727})$.

We will first estimate the complexity of the brute-force search method. This method may test all the sources in $\mathcal{S}(\mathcal{D})$ to check if any of them satisfy a set of linear equations in (1). To test if each equation $\mathbf{y}_i = \mathbf{H}_i\mathbf{x}_i$ in (1) is satisfied, the following computations are needed:

- Since $\mathbf{H}_i \in \mathbb{F}^{m\times n}$, and $\mathbf{x}_i \in \mathbb{F}^n$, hence we need $m \times n$ multiplications plus $m \times (n-1)$ additions to compute $\mathbf{H}_i\mathbf{x}_i$.
- Because $\mathbf{y}_i \in \mathbb{F}^m$, at most m scalar comparisons may be implemented to test if $\mathbf{y}_i$ is equal to $\mathbf{H}_i\mathbf{x}_i$.

Hence it takes at most $[mn + m(n-1) + m] = 2mn$ operations to test if an n-tuples satisfies a linear equation in (1), which is bounded by $O(mn)$. Because there are actually s equations in (1), and the size of the source space is $|\mathbb{F}|^n|\mathcal{D}|$, hence the complexity of the brute-force method is bounded by $O(|\mathbb{F}|^n|\mathcal{D}|msn)$.

For our algorithm, we need an extra one-time preprocessing step, i.e,

- Generating the set: $\{(\tilde{\delta}_k, \mathbf{y'}_k)|\mathbf{y'}_k = \mathbf{P}\tilde{\delta}_k, \forall\ \tilde{\delta}_k \in \tilde{\mathcal{D}}, k = 1, \ldots, |\tilde{\mathcal{D}}|\}$. Similar to the previous analysis, its complexity is bounded by $O(rsn|\tilde{\mathcal{D}}|) = O(rsn|\mathcal{D}|)$ ($\mathbf{P} \in \mathbb{F}^{r\times sn}$).
- Sorting the above set by $\mathbf{y}'$: because the divide-and-conquer algorithm takes complexity $O(n\log n)$ (assume the complexity of basic comparison is $O(1)$) to sort a set, where n is the size of the set [10, chp. 4]. For our algorithm, the complexity of the basic comparison is $O(r)$ due to $\mathbf{y}' \in \mathbb{F}^r$, hence the complexity of sorting this set is $O(r|\mathcal{D}|\log(|\mathcal{D}|))$.
- Computing $(\mathbf{H^T H})^{-1}\mathbf{H^T}$ ($\mathbf{H} \in \mathbb{F}^{sm\times n}$) in (20): its complexity is bounded by $O(n^2sm)$.

So the overall complexity of the one-time preprocessing step is $O(rsn|\mathcal{D}| + r|\mathcal{D}|\log(|\mathcal{D}|) + n^2sm)$.

Next, we will analyze the decoding complexity.

- Solving (16): applying the binary search algorithm for searching the solution in the sorted table has the complexity of $O(r\log(|\mathcal{D}|))$ (The complexity of the binary search algorithm is $\log n$, and $\mathbf{y}' \in \mathbb{F}^r$).
- Substituting $(\hat{\mathbf{d}}_1, \ldots, \hat{\mathbf{d}}_s)$ into (20): $O(msn)$.
- Estimating $\hat{\mathbf{v}}$ from (20): $O(msn)$.
- Combining $(\hat{\mathbf{d}}_1, \ldots, \hat{\mathbf{d}}_s)$ and $\hat{\mathbf{v}}$ to get $(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_s)$ ((21)): $O(sn)$.

So the decoding complexity is $O(r\log(|\mathcal{D}|)+2msn+sn) = O(r\log(|\mathcal{D}|)+msn)$, which is only a fraction of that required by the brute-force search method $O(|\mathbb{F}|^n|\mathcal{D}|msn))$.

One may notice that our method needs to set up a $|\mathcal{D}|$-size table, which consumes an extra space. However, to make the $\mathbf{P}|_{\tilde{\mathcal{D}}}$ (cf. Theorem 1) to be invertible, $\mathcal{D}$ should not be larger than $|\mathbb{F}^r|$, where r is the length of each source data. Further, if we know the exact structure of $\mathcal{D}$, we may design a more optimized algorithm that maintains a good balance between space and speed. Take the special case of HCMSs for instance. Since only one non-zero scalar is contained in δ (cf. Example 1), hence for the first step of the algorithm, we need only to find which column of $\mathbf{P}$ corresponds with the nonzero element, whose complexity is $O(rsn)$ ($\mathbf{P}$ has the size of $r \times sn$). Therefore, the overall complexity for the HCMSs is $O(rsn+msn) = O(msn)$ $(m \geq r)$ which shows both good speed and memory saving.

V. Conclusion

This letter focuses on the decoding algorithm of Partion Matrix Codes which take Confined-Correlated Source with Deviation Symmetry as input. We start from the basic properties of that coding method, and obtain the analyzed solution for the source. Based on the results, we propose a two-step algorithm. For the first step, we set up a sorted table to search the solution in space $\mathcal{D}$, and the second step is to solve a linear equation. Finally, we analyze the computational complexity of our algorithm and compare it with the brute-force search method. The results indicate the efficiency of our algorithm.

In the future, we will design more codes beyond Generalized Hamming Sources and try to apply this code for the joint Gaussian distribution.

References

[1] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *Information Theory, IEEE Transactions on*, vol. 19, no. 4, pp. 471–480, 1973.

[2] S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (discus): design and construction," in *Data Compression Conference, 1999. Proceedings. DCC '99*, 1999, pp. 158–167.

[3] ——, "Generalized coset codes for distributed binning," *Information Theory, IEEE Transactions on*, vol. 51, no. 10, pp. 3457–3474, 2005.

[4] A. Wyner, "Recent results in the shannon theory," *Information Theory, IEEE Transactions on*, vol. 20, no. 1, pp. 2–10, 1974.

[5] J. Garcia-Frias and Y. Zhao, "Near-shannon/slepian-wolf performance for unknown correlated sources over awgn channels," *Communications, IEEE Transactions on*, vol. 53, no. 4, pp. 555–559, 2005.

[6] V. Stankovic, A. Liveris, Z. Xiong, and C. Georghiades, "On code design for the slepian-wolf problem and lossless multiterminal networks," *Information Theory, IEEE Transactions on*, vol. 52, no. 4, pp. 1495–1507, 2006.

[7] S. Cheng and R. Ma, "The non-existence of length-5 perfect slepian-wolf codes of three sources," in *Data Compression Conference (DCC), 2010*, 2010, pp. 528–528.

[8] R. Ma and S. Cheng, "The universality of generalized hamming code for multiple sources," *Communications, IEEE Transactions on*, vol. 59, no. 10, pp. 2641–2647, 2011.

[9] ——, "Zero-error slepian-wolf coding of confined correlated sources with deviation symmetry," *Arxiv preprint arXiv:1308.0632*, 2013.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[11] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *Proceedings of the 44th symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 887–898.