A Two-Step Screening Algorithm to Solve Linear Error Equations for Blind Identification of Block Codes Based on Binary Galois Field

Qian Liu[1*], Hao Zhang[1], Peidong Yu[1], Gang Wang[1], and Zhaoyang Qiu[1]
[1] School of Information System Engineering, Information Engineering University
Zhengzhou, Henan 450001 China
[e-mail: liuqian2006815@126.com]
[*]Corresponding author: Qian Liu

Received April 12, 2021; revised July 24, 2021; accepted August 25, 2021; published September 30, 2021

Abstract

Existing methods for blind identification of linear block codes without a candidate set are mainly built on the Gauss elimination process. However, the fault tolerance will fall short when the intercepted bit error rate (BER) is too high. To address this issue, we apply the reverse algebra approach and propose a novel "two-step-screening" algorithm by solving the linear error equations on the binary Galois field, or GF(2). In the first step, a recursive matrix partition is implemented to solve the system linear error equations where the coefficient matrix is constructed by the full codewords which come from the intercepted noisy bitstream. This process is repeated to derive all those possible parity-checks. In the second step, a check matrix constructed by the intercepted codewords is applied to find the correct parity-checks out of all possible parity-checks solutions. This novel "two-step-screening" algorithm can be used in different codes like Hamming codes, BCH codes, LDPC codes, and quasi-cyclic LDPC codes. The simulation results have shown that it can highly improve the fault tolerance ability compared to the existing Gauss elimination process-based algorithms.

Keywords: Two-step screening algorithm, blind identification, linear block code, linear error equation, matrix partition, Galois field, Gauss elimination process

This work was supported by the National Natural Science Foundation of China (No. 61802430, No. 62072057) and China Postdoctoral Science Foundation (No. 2016M603035). The authors would like to thank the reviewers for their insightful comments and helpful suggestions.

1. Introduction

Error-correcting codes [1] are used in telecommunication systems to correct errors induced by a noisy channel and increase the reliability of digital data transmission. The information blocks are transmitted into the encoder defined by a generator matrix or a parity-check matrix, and then outputs the block codewords based on the binary Galois field, which is defined as GF(2) (denoted by F_2).

In the context of a cooperative communication model, the receiver knows all those encoders which the transmitter uses, like the situation of adaptive modulation and coding (AMC) [2]. Therefore, the receiver can choose the correct encoder from a candidate set by blind identification of channel coding. The received codewords are then decoded to obtain the information. When dealing with blind identification of channel coding within a candidate set, researchers converted this problem into a maximum-likelihood problem. That is a detection and recognition problem, it is equivalent to finding the minimum distance between the incorrect codewords and correct codewords [3]. Alternatively, authors of [4] [5] [6] used the statistical hypothesis tests method to solve this problem. In order to improve the fault tolerance ability of those methods, some researchers utilized the soft information of the intercepted bitstream by defining functions such as the log-likelihood ratio (LLR) [7], the difference of likelihood [8], the cosine of syndrome a posterior probability (SPP) [9]. When those functions reach the maximum value, the member in the candidate set will be considered as the correct encoder.

While in a non-cooperative context, (for example, military applications or spectrum surveillance applications), a third party intercepts the signals transmitted between the two legal users and aims to retrieve helpful information. Suppose that this third party knows the parameters of the demodulation and scrambler if used. The adversary can only access the intercepted noisy binary stream exchanged between the legal users. For decoding the intercepted codewords to acquire helpful information, the adversary has to recover the corresponding generator matrix or parity-check matrix of the encoder scheme. Assuming that the starting bit of an entire codeword in the bitstream and the block length of the encoder are already available, we only need to focus on reconstructing the generator matrix or parity-check matrix of the encoder from the noisy bitstream without any prior knowledge being known.

In recent years, researchers have been devoted to solving the problem above. However, most of them applied BCH codes / RS codes with their cyclic structure [10] [11] or convolution codes / Turbo codes using their recursive property [12] [13] [14] [15]. However, seldom research worked on the general linear block codes without special structure. Furthermore, it becomes difficult when no candidate set is available, especially for long linear block codes. People usually implement the Gauss elimination method on the codeword matrix to derive dual codes. Research in [16] used a decision rule to obtain the correct sparse parity-checks from low-weight dual codewords. The derived parity-checks were used iteratively to decode the codewords with error. The study in [17] applied the Gauss elimination process to obtain the kernel space of the square codeword matrix, and then chose the correct parity-check vectors from them using some decision criteria. In [18], the work applied Gauss-Jordan elimination through pivoting (GJETP) to transform the noisy codewords matrix to an echelon one, and then proposed the “almost rank ratio criterion” to find dependent columns and derive the parameters and parity-checks of the encoder at the same time. The method proposed in [19] introduced a decoding technique approach to speed up the process of acquiring parity-checks based on the work in [18]. For the reconstruction of the parity-check matrices of LDPC codes, an algorithm [20] was proposed. Which needed much less iterations compared with [19] for using a technique BGCE (Bidirectional Gaussian Column Elimination). Most of the existing works based on Gaussian column elimination perform well when the bit error rate (BER) is low. However, as we know, if the intercepted bit stream corrupted by a high BER, those algorithms will be invalidated.

In this paper, we assume the framework synchronization, block length and rate of the encoder have already been known, because these parameters have been studied in-depth in [6] [18] [22] [23] [24] [25] [26]. In this situation we only devote our energy to reconstruct the parity-check matrix of linear block code from the noisy intercepted bit stream having a high BER. Inspired by the work in [21], we propose an algorithm based on linear algebra and matrix partition theory to solve the linear equations with error on F_2, our algorithm has a stronger fault tolerance ability than methods based on Gaussian column elimination.

The rest of this paper is organized as follows. In section 2, we introduce the notations and translate the reconstruction problem into algebraic equations. In section 3, we explain how to retrieve the parity-checks and give the whole concrete algorithm. Simulation results and analysis of the computational complexity of our algorithm are presented in section 4. Finally, we summarize our work in section 5.

2. Problem Description and the Algebraic Approach

2.1 The Related Mathematical Problem

We only study the problem of how to reconstruct the generator matrix or parity-check matrix from the intercepted bit stream. The whole flow chart of the operation which deals with blind identification of channel coding and decoding is displayed in Fig. 1.

Let $\boldsymbol{G}$ be a generator matrix and $\boldsymbol{H}$ be the corresponding parity-check matrix, they satisfy the orthogonal relationship in F_2 : $\boldsymbol{G}\boldsymbol{H}^{\mathrm{T}} = \mathbf{0}$. Where $\boldsymbol{H}^{\mathrm{T}}$ stands for the transposition of $\boldsymbol{H}$. The block code space generated by $\boldsymbol{G}$ is denoted as $\mathbb{C}$, and $\mathbb{C}^{\perp}$ represents the dual space spanned by the row vectors of $\boldsymbol{H}$. Let $\boldsymbol{m}_i = (m_{i1}, m_{i2}, \cdots, m_{ik})$ be the i-th information block, where $k = n\rho$, and the i-th codeword is $\boldsymbol{c}_i = (c_{i1}, c_{i2}, \cdots, c_{in}) = \boldsymbol{m}_i\boldsymbol{G}$, thus we have $\boldsymbol{c}_i\boldsymbol{H}^{\mathrm{T}} = \mathbf{0}$. Suppose codewords are sent to the binary symmetric channel (BSC) with cross-over probability P_e, $\boldsymbol{c}_i$ is the input and $\boldsymbol{a}_i = (a_{i1}, a_{i2}, \cdots, a_{in})$ $(i = 1, 2, \cdots, M)$ is the output, where M is the number of the total intercepted codewords, we have

$$a_{ij} = c_{ij} + e_{ij} \; (\; i = 1, 2, \cdots, M,\; j = 1, 2, \cdots, n), \tag{1}$$

where $e_{ij} \in \{0,1\}$ with $\Pr(e_{ij} = 1) = P_e$ and $\Pr(e_{ij} = 0) = 1 - P_e$.

Suppose N is some positive integer slightly larger than the code block length n, a codeword matrix $A = \left(a_1^{\mathrm{T}}, a_2^{\mathrm{T}}, \cdots, a_N^{\mathrm{T}}\right)^{\mathrm{T}}$ is then constructed by part of the output codewords which are derived from the encoded codeword matrix $\boldsymbol{C} = \left(\boldsymbol{c}_1^{\mathrm{T}}, \boldsymbol{c}_2^{\mathrm{T}}, \cdots, \boldsymbol{c}_N^{\mathrm{T}}\right)^{\mathrm{T}}$. Thus, we have $\boldsymbol{A} = \boldsymbol{C} + \boldsymbol{E}$, where $\boldsymbol{E} = \left(e_{ij}\right)_{N\times n}$. If $\boldsymbol{h}$ is a parity-check of $\mathbb{C}$, we have $\boldsymbol{C} \cdot \boldsymbol{h}^{\mathrm{T}} = \mathbf{0}$, while $\boldsymbol{A} \cdot \boldsymbol{h}^{\mathrm{T}} \neq \mathbf{0}$ in general.

We want to derive all the correct parity-checks $\boldsymbol{h}$ from $\boldsymbol{A}\cdot\boldsymbol{h}^{\mathrm{T}}\neq\boldsymbol{0}$ so that we can reconstruct the parity-check matrix $\boldsymbol{H}$.

2.2 The Algorithms Based on Gauss Column Elimination

Gallager gave the following proposition in [1]:

If a codeword of block length n is received after transmission through a BSC with cross-over probability P_e, the probability that the number of error bits is even is given as

$$\frac{1+\left(1-2P_e\right)^n}{2},$$

thus we have

$$\Pr(\boldsymbol{A}\cdot\boldsymbol{H}^{\mathrm{T}}\neq 0)=1-\Pr(\boldsymbol{A}\cdot\boldsymbol{H}^{\mathrm{T}}=0)=1-\left[\frac{1+\left(1-2Pe\right)^{w(\boldsymbol{h})}}{2}\right]^N, \tag{2}$$

where $w(\boldsymbol{h})$ is the Hamming weight of vector $\boldsymbol{h}$. Conversely, we can derive the conclusion : if a vector $\boldsymbol{x}$ makes $w\left(\boldsymbol{A}\cdot\boldsymbol{x}^{\mathrm{T}}\right)$ small, it is most likely to be a parity-check of $\mathbb{C}$.

From [20] we know, we can derive all the parity-checks through implying Gauss column elimination on the error-free codeword matrix. Based on this fact and the above conclusion, Authors in [18][19][20] obtain parity-checks by searching low weight columns (dependent columns) from the noisy codeword matrix after Gauss column elimination. However, may not all parity-checks can be found during the above step, thus a decoding process is introduced to correct the erroneous codewords to reduce the BER. Through this iterative process we can obtain all the parity-checks.

From [18] we know the method Gauss column elimination is seriously affected by error bits in codewords, because error bits can spread during column transformation. Therefore, the algorithms based on Gauss column elimination are ineffective when the BER is high. In this case, we must look for another method with strong fault tolerance to reconstruct the parity-check matrix. To overcome the shortcomings of error propagation, we take steps to resolve the following linear equations directly.

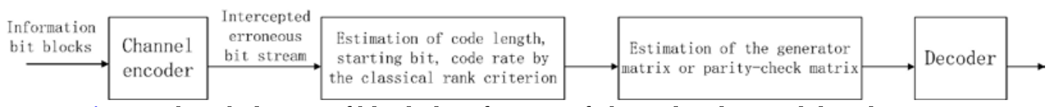
Information bit blocks
Channel encoder
Intercepted erroneous bit stream
Estimation of code length, starting bit, code rate by the classical rank criterion
Estimation of the generator matrix or parity-check matrix
Decoder

Fig. 1. The whole step of blind identification of channel coding and decoding

2.3 The Algorithm Based on Resolving Systems of Equations

In a noise-free context, there is $\boldsymbol{E}=0$. If we have enough codewords, the number of the valid equations in $\boldsymbol{AX}^{\mathrm{T}}=\boldsymbol{CX}^{\mathrm{T}}=\boldsymbol{0}$ is k, i.e. $\mathrm{rank}(\boldsymbol{A})=k$. We can, therefore, solve the linear equations

$$\boldsymbol{CX}^{\mathrm{T}}=\boldsymbol{0} \tag{3}$$

to derive a fundamental system of solutions. Due to the orthogonal relationship between codewords and parity-checks, we can derive $n-k$ linear independent vectors to retrieve the systematic parity-check matrix $\boldsymbol{H}_{(n-k)\times n}=\left(\boldsymbol{P}^{\mathrm{T}}\quad \boldsymbol{I}_{n-k}\right)$, where $\boldsymbol{I}$ is a unit matrix of $(n-k)\times(n-k)$, based on which the systematic generator matrix $\boldsymbol{G}_{k\times n} = \left(\boldsymbol{I}_k \quad \boldsymbol{P}\right)$ is retrieved.

However, in a noisy environment, the number of the valid equations in $\boldsymbol{AX}^{\mathrm{T}} = \mathbf{0}$ is much larger than k due to the existing of error bits in $\boldsymbol{A}$. Suppose $\operatorname{rank}\left(\boldsymbol{A}\right) = n$, we have to solve an *over-determined systems* of equations

$$\boldsymbol{AX}^{\mathrm{T}} = \mathbf{0}. \tag{4}$$

However, it is impossible to find a non-zero solution which satisfies all the equations in (4). Instead, what we want to do is to find solutions which satisfy the most equations in (4). This means we expect to find several solutions of the equations $\boldsymbol{AX}^{\mathrm{T}} = \boldsymbol{b}$, where $\boldsymbol{b}$ is a column vector belonging to F_2^M with $w(\boldsymbol{b}) = \varepsilon$. Where F_2^M is the extension field of F_2, and ε is some pre-set positive integer.

Since $w(\boldsymbol{Ax}^{\mathrm{T}}) = \varepsilon$ is equivalent to $w(\boldsymbol{xA}^{\mathrm{T}}) = \varepsilon$, by applying the linear equations $\boldsymbol{xA}^{\mathrm{T}} = \boldsymbol{b}^{\mathrm{T}}$ and $w(\boldsymbol{b}^{\mathrm{T}}) = \varepsilon$,we have

$$w(\boldsymbol{xB}) = \varepsilon, \tag{5}$$

Where $\boldsymbol{B} = \boldsymbol{A}^{\mathrm{T}}$ is a matrix of $n \times N$. The Definitions 2.1 and 2.2 in linear algebraic are described below:

Definition 2.1 Equation $w(\boldsymbol{xB} + \boldsymbol{b}) = \varepsilon$ is called a *linear error equations* [12], where $\boldsymbol{B}$ is a $n \times N$ matrix, $\boldsymbol{b}$ is a row vector of N dimension, N is larger than n, ε is an integer in the interval $\left[0, N\right]$.

Definition 2.2 Matrix $\boldsymbol{Q}$ is called a row (or column) permutation matrix, if $\boldsymbol{Q}$ can be represented by a product of many matrices, that is $\boldsymbol{Q} = \boldsymbol{Q}_1\boldsymbol{Q}_2\cdots\boldsymbol{Q}_s$, where $\boldsymbol{Q}_i$ $\left(i = 1, 2, \cdots, s\right)$ is an elementary row (or column) exchange matrix.

Property 2.1 For every permutation matrix $\boldsymbol{Q}$, we have $w\left(\boldsymbol{yQ}\right) = w\left(\boldsymbol{y}\right)$ and $\boldsymbol{Q}^{-1}$ is a permutation matrix, too.

Obviously, we have $\boldsymbol{b} = \mathbf{0}$ in (5). How to solve (5) in the changing process of the pre-set ε to obtain all the correct parity-checks?

Of course, we can solve equations (5) by randomly selecting n-dimensional vectors to obtain the vectors which make ε smaller than some threshold. These vectors are considered to be parity-checks of the encoder. However, the operation has a large computation and serious time delay when n is large (such as LDPC codes).

In the next section, we will propose an algorithm to derive solutions to the linear error equations (5) using a two-step method which don't need too much codewords. Actually, we derive a solution of the equations by dividing it into several sections and solving every section in turn.

3. Resolve the Linear Error Equations and Summarize the Algorithm

3.1 Solve the Linear Error Equations

The process of solving the linear error equations can be divided into two steps: I) recursive decomposition of the coefficient matrix $\boldsymbol{B}$ and II) iterative solution of the linear equations.

3.1.1 Recursive Decomposition of the Coefficient Matrix

At first, we give a lemma like that in [6] with a simple proof.

For a given coefficient matrix $\boldsymbol{B}$ in (5), we have

Lemma 3.1 Given a $n \times N$ $(N > n)$ matrix $\boldsymbol{B}$, there are an invertible $n \times n$ matrix $\boldsymbol{P}_1$ and a $N \times N$ permutation matrix $\boldsymbol{Q}_1$ such that

$$\boldsymbol{P}_1 \boldsymbol{B} \boldsymbol{Q}_1 = \begin{pmatrix} \boldsymbol{I}_{r_1} & \boldsymbol{B}_1 \\ \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \text{ or } \boldsymbol{P}_1 \boldsymbol{B} \boldsymbol{Q}_1 = \begin{pmatrix} \boldsymbol{I}_n & \boldsymbol{B}_1 \end{pmatrix}.$$

Where r_1 is the rank of $\boldsymbol{B}$ in the first case and the second case is $r_1 = n$.

Similarly, we can take the same operation on $\boldsymbol{B}_1$ and derive an invertible $r_1 \times r_1$ matrix $\boldsymbol{P}_1$ and a $(N - r_1) \times (N - r_1)$ permutation matrix $\boldsymbol{Q}_1$, as well as a $r_2 \times (n - r_1 - r_2)$ matrix $\boldsymbol{B}_2$. Going on in this manner, we summarize this recursive process as follows:

For each $\boldsymbol{B}_i$ $(i = 0, 1, 2, \cdots, l)$, we have

$$\boldsymbol{B}_i = \boldsymbol{P}_{i+1}^{-1} \begin{pmatrix} \boldsymbol{I}_{r_{i+1}} & \boldsymbol{B}_{i+1} \\ \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \boldsymbol{Q}_{i+1}^{-1} \text{, if rank(} \boldsymbol{B}_i \text{)=} r_{i+1} < r_i, \tag{6}$$

or

$$\boldsymbol{B}_i = \boldsymbol{P}_{i+1}^{-1} \begin{pmatrix} \boldsymbol{I}_{r_{i+1}} & \boldsymbol{B}_{i+1} \end{pmatrix} \boldsymbol{Q}_{i+1}^{-1} \text{, if rank(} \boldsymbol{B}_i \text{)=} r_{i+1} = r_i. \tag{7}$$

Where $\boldsymbol{B}_0 = \boldsymbol{B}$, $r_0 = n$, $N_0 = N$, $\boldsymbol{B}_i$ is a $r_i \times N_i$ matrix, $\boldsymbol{P}_{i+1}$ is a $r_i \times r_i$ non-singular matrix, $\boldsymbol{Q}_{i+1}$ is a $N_i \times N_i$ permutation matrix, $\boldsymbol{I}_{r_{i+1}}$ is an identity matrix of $r_{i+1} \times r_{i+1}$, the rank of $\boldsymbol{B}_i$ is r_{i+1}, and there is $N_i = N_{i+1} + r_{i+1}$.

When does this recursive decomposition process end? Namely, how to determine l? If $\boldsymbol{B}_{l+1} = \boldsymbol{0}$ or $N_l = r_{l+1}$, the recursive procedure terminates. The recursive decomposition of the coefficient matrix $\boldsymbol{B}$ can be organized in Algorithm 1.

Algorithm 1

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Inputs: a $N \times n$ codeword matrix $\boldsymbol{A}$
Outputs: $r_1, r_2, \cdots, r_{l+1}$ and $\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_{l+1}$
$\boldsymbol{B} = \boldsymbol{A}^{\mathrm{T}}$;
$N_0 = N$;
$i = 1$;
Do
find $\boldsymbol{P}_1, \boldsymbol{Q}_1$ such that $\boldsymbol{P}_1 \boldsymbol{B} \boldsymbol{Q}_1 = \begin{pmatrix} \boldsymbol{I}_{r_1} & \boldsymbol{B}_1 \\ \boldsymbol{O} & \boldsymbol{O} \end{pmatrix}$;
$r(i)$=rank$(\boldsymbol{B})$;
$\boldsymbol{P}(i) = \boldsymbol{P}_1$;
$i = i + 1$;
$\boldsymbol{B} = \boldsymbol{B}_1$;
While rank$(\boldsymbol{B}_1) \neq 0$
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3.1.2 Iterative Solution of the Linear Equations

First, the below Definition 3.1 is given:

Definition 3.1 The operation $\mathrm{T}_k(\boldsymbol{V})$ represents taking the first k columns of a matrix $\boldsymbol{V} = (v_1, v_2, \cdots, v_m)$ $(m \geq k)$. Namely, $\mathrm{T}_k(\boldsymbol{V}) = (v_1, v_2, \cdots, v_k)$.We then give a theorem like [12] with a simple proof.

Theorem 3.1 Under the decomposition of $\boldsymbol{B}_l$ $(l=0,1,2,\cdots)$ mentioned above, we have

$$w(\boldsymbol{xB})=w(\boldsymbol{x}_1)+w(\boldsymbol{x}_2)+\cdots+w(\boldsymbol{x}_{l+1}\boldsymbol{B}_{l+1}), \tag{8}$$

where $\boldsymbol{x}_l$ is a row vector of dimension r_l , and has the relationship with $\boldsymbol{x}_{l+1}$ as $\boldsymbol{x}_{l+1}=\mathrm{T}_{l+1}\left(\boldsymbol{x}_l\boldsymbol{P}_{l+1}^{-1}\right)$.

Proof We only prove the first step of the following proof, and the rest can be achieved in a similar way.

i) If there exist an $n\times n$ invertible matrix $\boldsymbol{P}_1$ and a $N\times N$ permutation matrix $\boldsymbol{Q}_1$ such that $\boldsymbol{B}=\boldsymbol{P}_1^{-1}\begin{pmatrix}\boldsymbol{I}_{r_1} & \boldsymbol{B}_1\\ \boldsymbol{O} & \boldsymbol{O}\end{pmatrix}\boldsymbol{Q}_1^{-1}$, where r_1 is the rank of $\boldsymbol{B}$, and $\boldsymbol{B}_1$ is a $r_1\times(N-r_1)$ matrix, then we have

$$\boldsymbol{xBQ}_1=\boldsymbol{xP}_1^{-1}\begin{pmatrix}\boldsymbol{I}_{r_1} & \boldsymbol{B}_1\\ \boldsymbol{O} & \boldsymbol{O}\end{pmatrix}\boldsymbol{Q}_1^{-1}\boldsymbol{Q}_1=\boldsymbol{xP}_1^{-1}\begin{pmatrix}\boldsymbol{I}_{r_1} & \boldsymbol{B}_1\\ \boldsymbol{O} & \boldsymbol{O}\end{pmatrix}=\left(\boldsymbol{x}_1,\boldsymbol{y}_1\right)\begin{pmatrix}\boldsymbol{I}_{r_1} & \boldsymbol{B}_1\\ \boldsymbol{O} & \boldsymbol{O}\end{pmatrix}=\left(\boldsymbol{x}_1,\boldsymbol{x}_1\boldsymbol{B}_1\right).$$

Where $\left(\boldsymbol{x}_1,\boldsymbol{y}_1\right)=\boldsymbol{xP}_1^{-1}$ and $\boldsymbol{x}_1=\mathrm{T}_{r_1}\left(\boldsymbol{xP}_1^{-1}\right)$. Thus, we have

$$w\left(\boldsymbol{xB}\right)=w\left(\boldsymbol{xBQ}_1\right)=w\left(\left(\boldsymbol{x}_1,\boldsymbol{x}_1\boldsymbol{B}_1\right)\right)=w\left(\boldsymbol{x}_1\right)+w\left(\boldsymbol{x}_1\boldsymbol{B}_1\right).$$

ii) If there exist a $n\times n$ non-singular matrix $\boldsymbol{P}_1$ and a $N\times N$ permutation matrix $\boldsymbol{Q}_1$ so that $\boldsymbol{B}=\boldsymbol{P}_1^{-1}\left(\boldsymbol{I}_{r_1}\quad \boldsymbol{B}_1\right)\boldsymbol{Q}_1^{-1}$, in this case, $r_1=\mathrm{rank}\left(\boldsymbol{B}\right)=r_0=n$, $\boldsymbol{B}_1$ is a $r_1\times(N-r_1)$ matrix, then we have

$$\boldsymbol{xBQ}_1=\boldsymbol{xP}_1^{-1}\left(\boldsymbol{I}_{r_1}\quad \boldsymbol{B}_1\right)\boldsymbol{Q}_1^{-1}\boldsymbol{Q}_1=\boldsymbol{xP}_1^{-1}\left(\boldsymbol{I}_{r_1}\quad \boldsymbol{B}_1\right)=\left(\boldsymbol{xP}_1^{-1},\boldsymbol{xP}_1^{-1}\boldsymbol{B}_1\right)=\left(\boldsymbol{x}_1,\boldsymbol{x}_1\boldsymbol{B}_1\right).$$

Here $\boldsymbol{x}_1=\boldsymbol{xP}_1^{-1}$, which can still be expressed by $\boldsymbol{x}_1=\mathrm{T}_{r_1}\left(\boldsymbol{xP}_1^{-1}\right)$. Therefore, we still have

$$w\left(\boldsymbol{xB}\right)=w\left(\boldsymbol{xBQ}_1\right)=w\left(\left(\boldsymbol{x}_1,\boldsymbol{x}_1\boldsymbol{B}_1\right)\right)=w\left(\boldsymbol{x}_1\right)+w\left(\boldsymbol{x}_1\boldsymbol{B}_1\right).$$

The following is similar to what we have done above. Suppose the total number of steps we can do is $l+1$. For each among the total $l+1$ steps, we always have $w\left(\boldsymbol{x}_i\boldsymbol{B}_i\right)=w\left(\boldsymbol{x}_{i+1}\right)+w\left(\boldsymbol{x}_{i+1}\boldsymbol{B}_{i+1}\right)$ $(i=0,1,2,\cdots,l)$ no matter which case it falls into. Therefore, Eq. (8) can be derived straightforwardly.

In the last step, there are two results: one is $\boldsymbol{B}_{l+1}=0$, i.e.,

$$\boldsymbol{B}_l=\boldsymbol{P}_{l+1}^{-1}\begin{pmatrix}\boldsymbol{I}_{r_{l+1}} & \boldsymbol{O}\\ \boldsymbol{O} & \boldsymbol{O}\end{pmatrix}\boldsymbol{Q}_{l+1}^{-1}\ ,$$

the other is $N_l=r_{l+1}$, i.e.,

$$\boldsymbol{B}_l=\boldsymbol{P}_{l+1}^{-1}\begin{pmatrix}\boldsymbol{I}_{r_{l+1}}\\ \boldsymbol{O}\end{pmatrix}\boldsymbol{Q}_{l+1}^{-1}.$$

Both of the two cases, however, can draw the same conclusion as

$$w(\boldsymbol{xB})=w(\boldsymbol{x}_1)+w(\boldsymbol{x}_2)+\cdots+w(\boldsymbol{x}_{l+1}).$$

Therefore, in the process of solving the systematic linear error equations (5), there exist finite row vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{l+1}$ satisfying

$$\varepsilon = w(\boldsymbol{x}_1) + w(\boldsymbol{x}_2) + \cdots + w(\boldsymbol{x}_{l+1}),\ 0 \le w(x_i) \le r_i,\ 1 \le i \le l+1. \tag{9}$$

Furthermore, we know $n = r_0 \ge r_1 \ge r_2 \ge \cdots \ge r_{l+1}$ as well as the resolution of the linear error equations depending on the decomposition of $\boldsymbol{B}$ completely. We present the iterative solution of the linear error equations in Algorithm 2.

Algorithm 2

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Input: $l$, $r_1, r_2, \cdots, r_{l+1}$ and $\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_{l+1}$;
    The number of iterations $T$;
Output: the set $X$ of solutions of (5)
$X = \varnothing$;
for $i$=1:$T$
  for $\varepsilon$=1:$\lfloor N/2 \rfloor$
    for $t$=1:$\binom{N}{\varepsilon}$
      split $\varepsilon$ to be $\varepsilon_1 + \varepsilon_2 + \cdots + \varepsilon_{l+1}$;
           randomly choose $x_{l+1}$ such that $w(x_{l+1}) = \varepsilon_{l+1}$;
      for $j$=$l$:1
        for repeat=1:100
        randomly generate $N_j - r_{j+1}$-dimensional vector $y_j$;
        if $w[(x_j, y_j)P_j] = \varepsilon_{j-1}$
          $x_{j-1} = (x_j, y_j)P_j$;
        else
          break
        end
          end
      end
                  $x = x_0$;
                  $X = X \bigcup \{x\}$;
     end
   end
 end
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3.1.3 A Simple Example

A simple example is provided below to describe the above process. For (6, 3) linear block code, whose generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix},$$

we received 9 codewords from the BSC with cross-over probability $P_e = 0.2$, which are $\boldsymbol{c}_1 = (0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0)$, $\boldsymbol{c}_2 = (1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1)$, $\boldsymbol{c}_3 = (0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1)$, $\boldsymbol{c}_4 = (1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1)$, $\boldsymbol{c}_5 = (0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1)$, $\boldsymbol{c}_6 = (1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0)$,

$\boldsymbol{c}_7 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$, $\boldsymbol{c}_8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$, $\boldsymbol{c}_9 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$, respectively. Thus, we have

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \text{ and } \boldsymbol{B} = \boldsymbol{A}^{\mathrm{T}} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix},$$

And to implement the recursive matrix partition on $\boldsymbol{B}$, we have

$$\boldsymbol{P}_1\boldsymbol{B}\boldsymbol{Q}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix},$$

where

$$\boldsymbol{P}_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}, \boldsymbol{B}_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \text{ and } \boldsymbol{P}_2\boldsymbol{B}_1\boldsymbol{Q}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

where

$$\boldsymbol{P}_2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } r_1 = 6,\ r_2 = 3,\ \boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{P}_1,\ \boldsymbol{x}_2 = \mathrm{T}_3\left(\boldsymbol{x}_1\boldsymbol{P}_2^{-1}\right).$$

When $\varepsilon = 1$, we only have $w(\boldsymbol{x}_1) = 1$ and $w(\boldsymbol{x}_2) = 0$. Thus $\boldsymbol{x}_2$ must be $(0,0,0)$, $\boldsymbol{y}_2$ has several different possibilities: $(1,0,0)$, $(0,1,0)$, $(0,0,1)$, $(1,1,0)$, $(1,0,1)$, $(0,1,1)$ and

$(1,1,1)$, we then have $\boldsymbol{x}_1 = (\boldsymbol{x}_2\ \boldsymbol{y}_2)\boldsymbol{P}_2$.

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,1,0,0)$, we have $\boldsymbol{x}_1 = (1,0,0,0,0,0)$, which satisfies $w(\boldsymbol{x}_1) = 1$, then

$$\boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{P}_1 = (1,0,1,0,1,0).$$

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,0,1,0)$, we have $\boldsymbol{x}_1 = (0,1,1,1,1,0)$, which does not satisfy $w(\boldsymbol{x}_1) = 1$.

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,0,0,1)$, we have $\boldsymbol{x}_1 = (0,0,0,0,0,1)$, which satisfies $w(\boldsymbol{x}_1) = 1$, then

$$\boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{P}_1 = (1,1,0,1,0,0).$$

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,1,1,0)$, we have $\boldsymbol{x}_1 = (1,1,1,1,1,0)$, which does not satisfy $w(\boldsymbol{x}_1) = 1$.

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,1,0,1)$, we have $\boldsymbol{x}_1 = (1,0,0,0,0,1)$, which does not satisfy $w(\boldsymbol{x}_1) = 1$.

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,0,1,1)$, we have $\boldsymbol{x}_1 = (0,1,1,1,1,1)$, which does not satisfy $w(\boldsymbol{x}_1) = 1$.

If $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,1,1,1)$, we have $\boldsymbol{x}_1 = (1,1,1,1,1,1)$, which does not satisfy $w(\boldsymbol{x}_1) = 1$.

When $\varepsilon = 2$, we have $w(\boldsymbol{x}_1) = 2$, $w(\boldsymbol{x}_2) = 0$ or $w(\boldsymbol{x}_1) = 1$, $w(\boldsymbol{x}_2) = 1$. In the first case, where $\boldsymbol{x}_2 = (0,0,0)$, when $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (0,0,0,1,0,1)$, there is $\boldsymbol{x}_1 = (1,0,0,0,0,1)$ satisfying $w(\boldsymbol{x}_1) = 2$ and $\boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{P}_1 = (0,1,1,1,1,0)$. In the second case, only when $(\boldsymbol{x}_2\ \boldsymbol{y}_2) = (1,0,0,0,0,0)$ or $(0,0,1,0,0,0)$ there are $\boldsymbol{x}_1 = (0,0,0,1,0,0)$, $\boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{P}_1 = (1,1,0,1,1,1)$ or $\boldsymbol{x}_1 = (0,0,1,0,0,0)$, $\boldsymbol{x} = \boldsymbol{x}_1\boldsymbol{P}_1 = (0,0,0,1,0,1)$, respectively.

The rest can be achieved in a similar way. So far, we have derived the solutions $(1,0,1,0,1,0)$, $(1,1,0,1,0,0)$ of (5) for $\varepsilon = 1$ and $(1,1,0,1,1,1)$, $(0,0,0,1,0,1)$ for $\varepsilon = 2$. Which can be repeated by intercepting another group of erroneous codewords to derive more solutions.

3.2 The Reconstruction of the Parity-Check Matrices of Linear Block Codes

Based on Gallager's conclusion [1], it is explored by Chabot [8] that, for a BSC with cross-over probability P_e, a codeword $\boldsymbol{c}$ is input to the channel and $\boldsymbol{a}$ is output. For a given vector $\boldsymbol{h} \in F_2^n$ (F_2^n is the extension field of F_2)and $\boldsymbol{A}$ being a $N \times n$ codewords matrix constructed by part of the output codewords, if $\boldsymbol{h}$ belongs to $\mathbb{C}^{\perp}$, then we have

$$\Pr\left(\boldsymbol{h}\cdot\boldsymbol{a}^{\mathrm{T}} = 0\right) = \frac{1+\left(1-2P_e\right)^{w(\boldsymbol{h})}}{2} \tag{10}$$

and

$$\Pr\left(\boldsymbol{h}\cdot\boldsymbol{a}^{\mathrm{T}} = 1\right) = \frac{1-\left(1-2P_e\right)^{w(\boldsymbol{h})}}{2}. \tag{11}$$

Thus, we can compute

$$\Pr\left(w\left(\boldsymbol{h}\boldsymbol{A}^{\mathrm{T}}\right) = \varepsilon \,\middle|\, \boldsymbol{h} \in \mathbb{C}^{\perp}\right) = \binom{M}{\varepsilon}\left[\frac{1+\left(1-2P_e\right)^{w(\boldsymbol{h})}}{2}\right]^{\varepsilon}\left[\frac{1-\left(1-2P_e\right)^{w(\boldsymbol{h})}}{2}\right]^{M-\varepsilon}.$$

If $\boldsymbol{h} \notin \mathbb{C}^{\perp}$, then there is

$$\Pr\left(\boldsymbol{h}\cdot\boldsymbol{a}^{\mathrm{T}} = 1\right) = \Pr\left(\boldsymbol{h}\cdot\boldsymbol{a}^{\mathrm{T}} = 0\right) = \frac{1}{2}, \tag{12}$$

and we have $\Pr\left(w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon \middle| \boldsymbol{h}\notin\mathbb{C}^{\perp}\right)=\binom{M}{\varepsilon}\left(\frac{1}{2}\right)^{M}$.

Therefore, $w\left(\boldsymbol{hA}^{\mathrm{T}}\right)$ obeys binomial distribution with different parameters according to $\boldsymbol{h}\in\mathbb{C}^{\perp}$ or not. When M is big enough, if $\boldsymbol{h}\in\mathbb{C}^{\perp}$, there is

$$w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=w\left(\boldsymbol{hB}\right)\approx\frac{M}{2}\left(1-\left(1-2P_e\right)^{w(\boldsymbol{h})}\right), \tag{13}$$

and if $\boldsymbol{h}\notin\mathbb{C}^{\perp}$, we have

$$w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=w\left(\boldsymbol{hB}\right)\approx\frac{M}{2}. \tag{14}$$

Thus, from the central limit theorem we know $w\left(\boldsymbol{hA}^{\mathrm{T}}\right)$ obeys normal distribution

$$\mathrm{N}\left(\frac{M\left[1+\left(1-2P_e\right)^{w(\boldsymbol{h})}\right]}{2},\frac{M\left[1+\left(1-2P_e\right)^{2w(\boldsymbol{h})}\right]}{4}\right)\text{ or }\mathrm{N}\left(\frac{M}{2},\frac{M}{4}\right)$$

with respect to $\boldsymbol{h}\in\mathbb{C}^{\perp}$ or $\boldsymbol{h}\notin\mathbb{C}^{\perp}$, respectively. Moreover, the difference of the value between (13) and (14) increases as *M* increases.

We can take advantage of the above conclusion in the reverse way. For an intercepted $M\times n$ codewords matrix $\boldsymbol{A}$, if there is a vector $\boldsymbol{x}\in F_2^n$ which makes $w\left(\boldsymbol{x}\cdot\boldsymbol{A}^{\mathrm{T}}\right)$ equal to some given ε, what is the probability of $\boldsymbol{x}\in\mathbb{C}^{\perp}$? That is to say, we want to compute $\Pr\left(\boldsymbol{x}\in\mathbb{C}^{\perp}\middle|w\left(\boldsymbol{xA}^{\mathrm{T}}\right)=\varepsilon\right)$ and to derive some ε which makes $\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\middle|w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\right)$ higher furthermore.

Based on the Bayes formula, we have

$$\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\middle|w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\right)=\frac{\Pr\left(w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\middle|\boldsymbol{h}\in\mathbb{C}^{\perp}\right)\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\right)}{\Pr\left(w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\middle|\boldsymbol{h}\in\mathrm{C}^{\perp}\right)\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\right)+\Pr\left(w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\middle|\boldsymbol{h}\notin\mathrm{C}^{\perp}\right)\Pr\left(\boldsymbol{h}\notin\mathbb{C}^{\perp}\right)}. \tag{15}$$

Since $\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\right)=\frac{2^{n(1-\rho)}}{2^n}$ and $\Pr\left(\boldsymbol{h}\notin\mathbb{C}^{\perp}\right)=1-\frac{2^{n(1-\rho)}}{2^n}$, we have

$$\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\middle|w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\right)=\frac{1}{\frac{2^k-1}{\left[1+\left(1-2P_e\right)^{w(\boldsymbol{h})}\right]^M}\cdot\left(\frac{1+\left(1-2P_e\right)^{w(\boldsymbol{h})}}{1-\left(1-2P_e\right)^{w(\boldsymbol{h})}}\right)^{\varepsilon}+1}, \tag{16}$$

which indicates that the probability $\Pr\left(\boldsymbol{h}\in\mathbb{C}^{\perp}\middle|w\left(\boldsymbol{hA}^{\mathrm{T}}\right)=\varepsilon\right)$ decreases as ε increases. This means, when a vector $\boldsymbol{x}$ makes $w\left(\boldsymbol{x}\cdot\boldsymbol{A}^{\mathrm{T}}\right)$ much smaller than $\frac{M}{2}$, possibly there is $\boldsymbol{x}\in\mathbb{C}^{\perp}$. Thus, we can resolve (5) for some small positive integer ε to derive the solutions which are probable to be parity-checks. The setting of ε will be presented in remark 1.

We have to choose the correct parity-checks from the derived solutions which are treated as possible parity-checks. Eq. (13) and (14) indicate that a number can be chosen as the threshold with a low false-alarm probability from the interval

$$\left[\frac{M\left[1-\left(1-2P_e\right)^{w(x)}\right]}{2}+3\sqrt{\frac{M\left[1-\left(1-2P_e\right)^{2w(x)}\right]}{4}},\frac{M}{2}-3\sqrt{\frac{M}{4}}\right] \tag{17}$$

according to the "3 standard deviation" if M is big enough.

In fact, if incorrect parity-checks are wrongly thought to be correct, it will heavily influent the reconstruction of generator matrix or parity-check matrix. To address this, a threshold β is set to decide whether $\boldsymbol{h}$ is a parity-check or not with the false-alarm probability given as

$$\Pr\left\{w\left(\boldsymbol{h}\boldsymbol{A}^{\mathrm{T}}\right)<\beta \middle| \boldsymbol{h}\notin\mathbb{C}^{\perp}\right\}=\int_{-\infty}^{\beta}\frac{1}{\sqrt{\pi M/2}}e^{\frac{(x-M/2)^2}{M/2}}\mathrm{d}x .$$

This indicates that the smaller β, the higher the false-alarm probability. In our method, it is empirically set to be $\beta=\frac{M}{2}\times 0.5$ with a false-alarm probability smaller than 0.3%. Let $\boldsymbol{A}_{check}$ be the codewords matrix with a dimension of $M\times n$, which is composed of M intercepted codewords and used to choose correct parity-checks. In general, the larger M is, the higher the probability of selecting the correct parity-checks [17]. Therefore, we choose $M=10n$ by the rule of thumb.

3.3 The Specific "Two-Step Screening" Blind Recognition Algorithm for Block Codes

Our algorithm is described in the sequel. The parameters used in our algorithm are defined in **Table 1**.

Table 1. The parameters in our algorithm

n	code block length
k	code dimension
M	the total number of the intercepted codewords
$\boldsymbol{A}_{check}$	A matrix is constructed by all the intercepted codewords
IT	the given number of iterations
β	the threshold used to choose correct solutions
Θ	The set of parity-checks

Suppose the input is the transpose of the received codewords $\boldsymbol{B}=\boldsymbol{A}^{\mathrm{T}}$ of a $n\times N$ matrix, whose columns are the received codewords. And the output is the parity-check matrix $\boldsymbol{H}$ or the systematic generator matrix $\boldsymbol{G}$. Our Algorithm 3 for the whole procedure is organized as follows:

Algorithm 3

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
$\Theta=\varnothing$;
for iter =1: $IT$
    $X=\varnothing$;
    randomly choose $N$ codewords from the total $M$ codewords to construct $\boldsymbol{A}$;
    $\boldsymbol{B}=\boldsymbol{A}^{\mathrm{T}}$;
    imply algorithm 1 on B;
    imply algorithm 2;
```

```
  for all $\boldsymbol{x} \in X$
        if $w(\boldsymbol{A}_{check} \cdot \boldsymbol{x}) \le \beta$
        $\Theta = \Theta \bigcup \{\boldsymbol{x}\}$;
        end
  end
    if $\text{rank}(\Theta) = n-k$
    break
    end
end
```
~~~~~~

If we can derive $n-k$ linear independent members from Θ to construct a $(n-k)\times n$ matrix, we may change the matrix to the systematic parity-check matrix $\boldsymbol{H} = (\boldsymbol{P}^{\mathrm{T}} \ \boldsymbol{I})$ by the elementary row transformation. It is easy to derive the generator matrix $\boldsymbol{G} = (\boldsymbol{I} \ \ \boldsymbol{P})$. If we cannot get $n-k$ linear independent parity-checks in a setting time T , it is failed to reconstruct the block code.

Below are five remarks for our algorithm:

Remark 1 In order to speed up derivation of parity-checks, we can pre-set the weight of the objective vectors $\boldsymbol{x}$ to be ω, then $\varepsilon = w(\boldsymbol{x}\boldsymbol{A}^{\mathrm{T}})$ will locate in interval (16) with a probability of more than 99.7% according to the "3 standard deviation". Thus ε can be chosen from interval (16). In turn, if we want to derive the parity checks with certain Hamming weight, we can choose ε from (16). This means there is no need to traverse all the integers within $[1, N]$. Especially when the block length of the codeword is small, we can choose small positive ε from (16), which simplifies the step of integer splitting of ε .

Remark 2 Our algorithm can be highlighted that it is suitable for LDPC codes and especially for quasi-cyclic LDPC codes because of the sparsity of their parity-check matrices. In fact, according to (10) $\varepsilon = w(\boldsymbol{x}\boldsymbol{A}^{\mathrm{T}}) \approx \frac{M}{2}\left(1-(1-2P_e)^{w(x)}\right)$ decreases as $w(\boldsymbol{x})$ decreases. Meanwhile, with the smallest weight of the parity-checks of LDPC codes, we can resolve (5) for small ε to derive all the vectors $\boldsymbol{x}$ with low Hamming weight, some of which are the sparse parity-checks direct. In case of quasi-cyclic LDPC codes, for a $m_b \times n_b$ basic parity-check matrix $\boldsymbol{H}_b$, we only need m_b parity-checks which belong to different sub-matrices to reconstruct the sparse quasi-cyclic parity-check matrix $\boldsymbol{H}$. This eliminates the need of the computationally expensive step to make parity-check matrix sparse like what the algorithm in [19] does.

Remark 3 When implementing the recursive matrix partition on the intercepted codeword matrix $\boldsymbol{B}$, we derive $\boldsymbol{B} = \boldsymbol{P}_1^{-1}\begin{pmatrix} \boldsymbol{I}_k & \boldsymbol{B}_1 \\ \boldsymbol{O} & \boldsymbol{O} \end{pmatrix}\boldsymbol{Q}_1^{-1}$ after the first step if $\boldsymbol{B}$ is error-free. $\boldsymbol{P}_1$ is the row transformation matrix which records the linear combination of the rows of $\boldsymbol{B}$ so that all the last $n-k$ rows of $\boldsymbol{P}_1$ are orthogonal with the columns of $\boldsymbol{B}$. Therefore, they are the total linear independent parity-check vectors. The parity-check vectors are, thus, completely linear independent.

Remark 4 When the integer ε is split into several non-negative integers $\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_k$, we have to point it out that, if $\varepsilon_i = 0 \ (1 \le i < k)$, then $\varepsilon_{i+1} = \varepsilon_{i+2} = \cdots = \varepsilon_k = 0$. In fact, we only
~~~~~~

derive $\boldsymbol{x}_i = \boldsymbol{0}$ from $\varepsilon_i = w(\boldsymbol{x}_i) = 0$. Meanwhile, we can derive that $\boldsymbol{x}_{i+1}$ is none other than $\boldsymbol{0}$ because $(\boldsymbol{x}_{i+1}, \boldsymbol{y}_{i+1})\boldsymbol{P}_{i+1}^{-1} = \boldsymbol{x}_i$ and $\boldsymbol{P}_{i+1}^{-1}$ has an inverse matrix, and the following is analogical. Therefore, this property simplifies the progress of solving these linear equations by excluding several situations of integer splitting.

Remark 5 In the simulation experiments, N is usually chosen to be slightly larger than n in each iteration so as to reduce the times of recursive matrix partition and thus computational cost, which is also an advantage of our algorithm.

4. The Computational Complexity and Simulation Analysis

In the following, the comparisons among our algorithm and other approaches described in [17] [18] and [19] which are based on Gauss elimination method about computational complexity are presented. We also show the recognition probability of these algorithms and our work for different kinds of linear block codes in simulation results.

4.1 Computational Complexity Analysis

At first, let us make a brief description of the algorithm in [17], [18], and [19].

In [17], a dual code method is used to recover the parameters of the encoder and acquire parity-checks. It applies the Gauss elimination process on the square matrix coming from the codeword matrix to obtain its kernel space, then applies a decision rule to the vectors in the kernel space to choose the correct parity-checks. When all the linear independent parity-checks are collected, the parity-check matrix is reconstructed.

In this process, suppose the iteration time is set to be T, the number of total codewords is M and the total number of the vectors solved in kernel space is M_1. There are totally $T(n-1)n^2 + M \cdot M_1 \cdot n - M_1$ addition and $M \cdot M_1 \cdot n$ multiplication needed. When the BER is high, the square codeword matrix is very likely to be full-rank and its kernel space is much likely to be an empty set. It is very difficult to derive enough solutions so that we cannot reconstruct the parity-check matrix in T times.

In [18], a rank-based method for identifying the parameters of the interleaver is given. It applies a Gauss-Jordan elimination through pivoting (GJETP) method to transform the constructed matrix to a lower triangular matrix which contains the almost dependent columns, then it calculates the almost rank ratio which depends on the number of the almost dependent columns. The minimum of the almost rank ratios corresponds to the correct parameters of the interleaver. This operation can retrieve the parity-checks from the almost dependent columns at the same time.

In this process, suppose the number of iterations is T, and the number of total codewords is M. We need add $\frac{T(n-1)nM}{2} + Tn(M-n-1)$ times and compare the values $T(n-1)$ times.

Algorithm in [19] is similar to the scheme in [18] except for introducing a decoding step to accelerate the process of parity-check matrix reconstruction. Therefore, on the basis of the computation of [18], $t \cdot w(h)$ multiplication and $t \cdot w(h) - 1$ comparison is added. Where $w(h)$ is the weight of the parity-check and t is the number of the derived parity-checks.

What follows is the computation of our algorithm. Suppose the number of iterations is T, and ε is changing from 1 to $\lfloor N/2 \rfloor$. When ε is expressed as the sum of several l nonnegative integer, suppose we try T_1 times. There are $T \cdot T_1 \cdot \lfloor N/2 \rfloor \cdot \sum_{i=0}^{l} (N_i + r_i)(r_i - 1)$ addition and $T \cdot \lfloor N/2 \rfloor \cdot T_1 \cdot \sum_{i=0}^{l} r_i^2$ multiplication needed.

From the above analysis, we can see that the computational amount of our algorithm is the biggest. However, when the BER is high, all the algorithms based on Gauss elimination are invalid. Our algorithm still works in this case.

4.2 Simulation Results

4.2.1 For Hamming Codes

We first take the examples of (7,4) and (15,11) Hamming codes, and set $N = 10$, T= 3 and $N = 20$, T=4, respectively. The one used for comparison is the algorithm in [17], which utilizes the Gauss elimination process to resolve the kernel space of the square codeword matrices and chooses the correct parity-checks from them. In **Fig. 2**, we can see that, for (7,4) Hamming code, our algorithm can reconstruct 100% of the generator matrix when BER=0.115, but it is only 0% for algorithm in [17] at the same time. For (15,11) Hamming code, the correct identification probability of algorithm is 100% when BER=0.058. However, it is also 0% for algorithm in [17] in the same case. Algorithm in [17] can reach a 100% correct identification probability when BER=0.005 for (15,11) Hamming code and when BER=0.009 for (7,4) Hamming code. Our algorithm has a strong fault tolerant ability.

In the course of parity-check matrix reconstruction for (7,4) Hamming code, M=300 and T=3. There are at most 6 times stochastic decomposition for each ε and 35 times traversal operation on the weight of the solution vectors of the equations in each cycle. Suppose the total number of solution vectors is $\tilde{M}$, then 10180 + 2099 $\tilde{M}$ additions and 12180 + 2100 $\tilde{M}$ multiplications are needed. When the method in [17] is used to solve the parity-checks under the same condition, the number of cycles is set to be 10. Suppose the total number of solution vectors is $\bar{M}$, then 2940 + 2099 $\bar{M}$ additions and 2100 $\bar{M}$ multiplications are needed. Therefore, the computational complexity of our scheme is higher than that of [17].

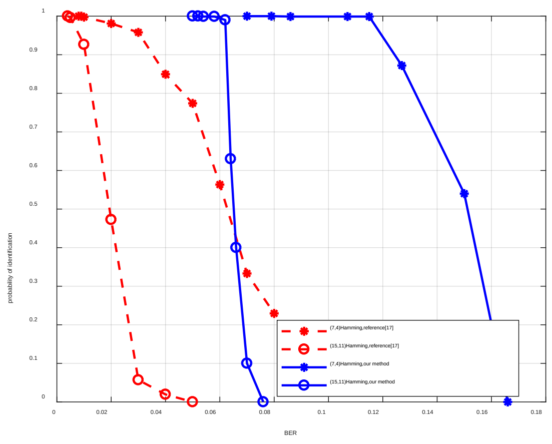

Fig. 2. Comparison of the probability of identification of (7,4) and (15,11) Hamming codes with respect to different BER between our algorithm and that in [17].

4.2.2 For BCH Codes

Let us take the (15,7,5) BCH codes as an example. Its generator polynomial is

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1,$$

and we set $N = 20$, $T = 4$. The comparison among our proposed scheme and those in [17] and [18] is shown in **Fig. 3**. We can see that the method in [18] has a better identification performance compared with that in [17] as the former finds "almost rank" instead of the rank of the codeword matrix. Its fault tolerance ability, however, is still limited by the bottleneck of the Gauss elimination process-based method. In this process, the correct identification probability of our algorithm is 100% when BER=0.11, but the algorithms in [17] and [18] can only reach 0% at the same time. The correct identification probability will reach 100% when BER=0.009 for algorithm in [17] and BER=0.05 for algorithm in [18], respectively. Therefore, the fault tolerance of our algorithm is at least one order of magnitude higher than the other two algorithms.

Due to the random choice of row vectors and the random error bits in the constructed square matrices, in [17], their kernel spaces are affected seriously. Only when the square matrices have no error bit, all the linear independent parity-check vectors can be found. Therefore, the results of this algorithm are not stable and its performance is not robust. Its fault tolerance is also the weakest.

In the course of parity-check matrix reconstruction for (15,7,5) BCH code, M=600 and T=4. There are at most 14 times stochastic decomposition for each ε and 1000 times traversal operation on the weight of the solution vectors of the equations in each cycle. Suppose the total number of solution vectors is $\tilde{M}$, then at most 923500 + 8400 $\tilde{M}$ additions and 1000000 + 9000 $\tilde{M}$ multiplications are needed. When the method in [18] is used, the number of cycles is set to 10. In total 719850 additions and 150 comparison operations are needed. It can be seen from the results that, although our scheme has the highest computational complexity among the three methods, it has the best fault tolerance ability.

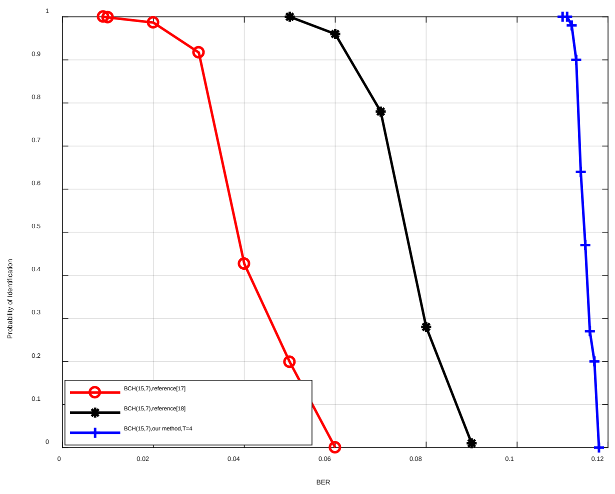

Fig. 3. Comparison among our algorithm and those in [17] and [18] for BCH codes.

4.2.3 For LDPC Codes

This sub-section is aimed to resolve all of the sparse parity checks of (31, 4, 4) LDPC code by using the proposed algorithm. We set $N = 40 \approx 1.3n$, $T = 30$. **Fig. 4** compares the probability of identification with respect to different BER by our method with that from [19]. Though algorithm in [19] has a better performance than algorithms in [17] and [18], we can still see that the fault tolerance of our algorithm is far more than algorithm in [19]. When BER=0.11, the correct identification probability of our algorithm is 100%, whereas that of [19] is 100% only when BER=0.057.

In our algorithm, for the purpose of saving computational cost, we resolve only the solutions which make $\varepsilon = w\left(\boldsymbol{hA}^{\mathrm{T}}\right)$ smallest as mentioned in remark 3. In the step of recursive matrix partition, it needs $0.6Tn^2\left(n-1\right)$ times addition operation, when resolving the linear error equations, it needs $2Tn^2$ times multiplication and $2Tn\left(n-1\right)$ times addition operation, and when choosing the correct parity-checks, it needs θMn times multiplication, $\theta\left(Mn-1\right)$ times addition and θ times comparison, where θ is the number of solutions resolved above. The computation of algorithm in [17] is about $10l \cdot n\left(n^2-1\right)+\dfrac{n^2\left(n-1\right)}{2}$ times addition and $10l \cdot n^3$ times multiplication, where l is the number of iterations. Because of the decoding step and calculation of almost rank instead of the rank, the computation of the algorithm in [19] is larger than that in [18], while the fault-tolerant ability of the former is better. Although the computation of our method is a little higher than the other approaches, it is able to deal with the identification problems when the cross-over probability of BSC is high.

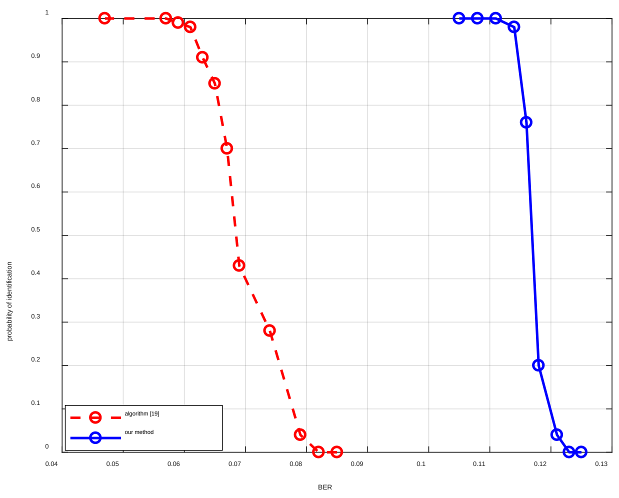

Fig. 4. Comparison of the probability of identification of (31,4,4) LDPC code with respect to different BER between our algorithm and that in [19].

4.2.4 For Quasi-Cyclic LDPC Codes

The computation time and complexity for acquiring parity- checks increase with the code length n. In order to reduce the computational cost of long codes, our method can be applied to channel coding with special structures.

For quasi-cyclic LDPC code, whose parity-check matrix $\boldsymbol{H}_{r\times n}$ consists of square sub-matrices of size $Z \times Z$, they are either zero or contain a cyclic-shifted identity matrix. $\boldsymbol{H}$ is represented by a base matrix $\boldsymbol{H}_b$ with m_b rows and n_b columns, here $m_b = \frac{r}{Z}$ and $n_b = \frac{n}{Z}$. Since the Hamming weights of the solutions we have derived are low, the lowest among them can be regarded as the row vectors of the sparse parity-check matrix $\boldsymbol{H}$. Thus, we only need to find m_b vectors in different sub-matrices to reconstruct $\boldsymbol{H}$. In fact, most of the parity check matrices of LDPC codes commonly used are quasi-cyclic.

Let us take the $(8Z, 4Z)$ LDPC code [26] [27] as an example, the parity check matrix is

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{P}^0 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}^0 & \boldsymbol{P}^1 & \boldsymbol{P}^0 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P}^1 & \boldsymbol{P}^4 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}^0 & \boldsymbol{P}^0 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P}^2 & \boldsymbol{0} & \boldsymbol{P}^6 & \boldsymbol{P}^0 & \boldsymbol{0} & \boldsymbol{P}^0 & \boldsymbol{P}^0 \\ \boldsymbol{P}^3 & \boldsymbol{0} & \boldsymbol{P}^5 & \boldsymbol{0} & \boldsymbol{P}^1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}^0 \end{pmatrix},$$

where $\boldsymbol{P}^0$ is a 16×16 unit matrix, $\boldsymbol{P}^i$ is the cyclic-shifted matrix of $\boldsymbol{P}^0$ to the right by i bits, and $\boldsymbol{0}$ is a 16×16 zero matrix.

When SNR=7 dB, $N = 150$, the iterations $T = 50$, in each iteration, there are 2000 times stochastic decomposition for each ε. Eighteen parity-checks are derived, the locations of whose nonzero bits are listed in **Table 2**.

In order to find correct Z, all of the common factors of $8Z$ and $4Z$ are traversed, and then the check code matrix is used to verify whether it is correct or not. Let take the first parity-check vector as an example, where the location of nonzero elements in it is 21, 40, 84, 100, respectively, denoted as $\boldsymbol{h}(21,\ 40,\ 84,\ 100)$ for convenience. The number of code words in the check matrix $\boldsymbol{A}_{check}$ is 1280, and we know the common factors of 128 and 64 are 2, 4, 8, 16, 32, 64. When $Z = 2$, the ones belonging to different sub-matrices are rotated to the right one bit each time and we can get $\boldsymbol{h}(22,\ 39,\ 83,\ 99)$. We have $w\left(\boldsymbol{A}_{check} \cdot \boldsymbol{h}^{\mathrm{T}}(21,40,84,100)\right) = 53$ but $w\left(\boldsymbol{A}_{check} \cdot \boldsymbol{h}^{\mathrm{T}}(22,39,83,99)\right) = 630$, which means $\boldsymbol{h}(22,\ 39,\ 83,\ 99)$ is not a parity-check vector, thus $Z \neq 2$. For $Z = 4,\ 8,\ 16$, the results are listed in **Table 3**, where we use $\boldsymbol{A}_c$ and $\boldsymbol{h}'$ to replace $\boldsymbol{A}_{check}$ and $\boldsymbol{h}^{\mathrm{T}}$ respectively for convenience. It is similar for $Z = 32,\ 64$, but we will not list them out.

We can see that all of $w\left(\boldsymbol{A}_{check} \cdot \boldsymbol{h}^{\mathrm{T}}\right)$ are much smaller than $\frac{M}{2} = 640$ only when $Z = 16$, which indicates $Z = 16$ is correct. In the following, let us reconstruct the sparse quasi-cyclic parity-check matrix by using the sparse parity checks we have derived from the linear error equations.

From $\boldsymbol{h}(21,\ 40,\ 84,\ 100)$ we can see that, the first module is **0**, the second module is $\boldsymbol{P}^4$, the third module is $\boldsymbol{P}^7$, the fourth and fifth module are **0**, the sixth and seventh module are P^3, and the eighth module is **0.** Thus, the first row of these modules is $\left(\boldsymbol{0}\ \boldsymbol{P}^1\ \boldsymbol{P}^4\ \ \boldsymbol{0}\ \boldsymbol{0}\ \boldsymbol{P}^0\ \ \boldsymbol{P}^0\ \ \boldsymbol{0}\right)$. For $\boldsymbol{h}(9,\ 43,\ 71,\ 118)$, the first module is $\boldsymbol{P}^8$, the second module is **0**, the third module is $\boldsymbol{P}^{10}$, the forth module is **0**, the fifth module is $\boldsymbol{P}^6$, the sixth and seventh module are **0**, the eighth module is $\boldsymbol{P}^5$, and therefore , the second row is $\left(\boldsymbol{P}^3\ \ \boldsymbol{0}\ \boldsymbol{P}^5\ \ \boldsymbol{0}\ \ \boldsymbol{P}^1\ \ \boldsymbol{0}\ \boldsymbol{0}\ \boldsymbol{P}^0\right)$. For $\boldsymbol{h}(3,\ 51,\ 68,\ 83)$, the first module is $\boldsymbol{P}^2$, the second and third modules are **0**, the forth module is $\boldsymbol{P}^2$, the fifth module is $\boldsymbol{P}^3$, the sixth module is $\boldsymbol{P}^2$, the seventh and eighth modules are **0**, resulting in the third row of modules $\left(\boldsymbol{P}^{\mathbf{0}}\ \mathbf{0}\ \mathbf{0}\ \boldsymbol{P}^{\mathbf{0}}\ \boldsymbol{P}^{\mathbf{1}}\ \boldsymbol{P}^{0}\ \mathbf{0}\ \mathbf{0}\right)$. For $\boldsymbol{h}(7,\ 28,\ 41,\ 64,69,74,106,116,122)$, which is the combination of $\boldsymbol{h}(7,\ 41,\ 69,\ 116)$ and $\boldsymbol{h}(28,\ 64,\ 74,\ 106,122)$, $\boldsymbol{h}(7,\ 41,\ 69,\ 116)$ belongs to $\left(\boldsymbol{P}^{3}\ \mathbf{0}\ \boldsymbol{P}^{\mathbf{5}}\ 0\ \ \boldsymbol{P}^{\mathbf{1}}\ \mathbf{0}\ \mathbf{0}\ \boldsymbol{P}^{0}\right)$ but $\boldsymbol{h}(28,\ 64,\ 74,\ 106,122)$ is not a member of them. In fact, for $\boldsymbol{h}(28,\ 64,\ 74,\ 106,122)$, the first module is **0**, the second module is $\boldsymbol{P}^{11}$, the third module is **0**, the forth module is $\boldsymbol{P}^{15}$, the fifth module is $\boldsymbol{P}^{9}$, the sixth module is **0**, the seventh and eighth modules are $\boldsymbol{P}^{9}$, and thus we get the forth row of modules $\left(\mathbf{0}\ \boldsymbol{P}^{\mathbf{2}}\ \mathbf{0}\ \ \boldsymbol{P}^{\mathbf{6}}\ \boldsymbol{P}^{\mathbf{0}}\ \mathbf{0}\ \boldsymbol{P}^{0}\ \ \boldsymbol{P}^{0}\ \right)$. Up to now, the parity-check matrix of $(8Z,4Z)$ LDPC code is reconstructed successfully.

The correct identification probability with respect to different BER of our algorithm under different values of T is given in **Fig. 5**. The algorithm for comparison is algorithm in [19]. When $T=50$ and BER=0.013, the correct identification probability of our algorithm is 100%, whereas it reaches 100% when $T=100$ and BER=0.015. The correct identification probability increasing with the value of T is consistent with our intuition. However, the correct identification probability of algorithm in [19] is less than 50% in this case.

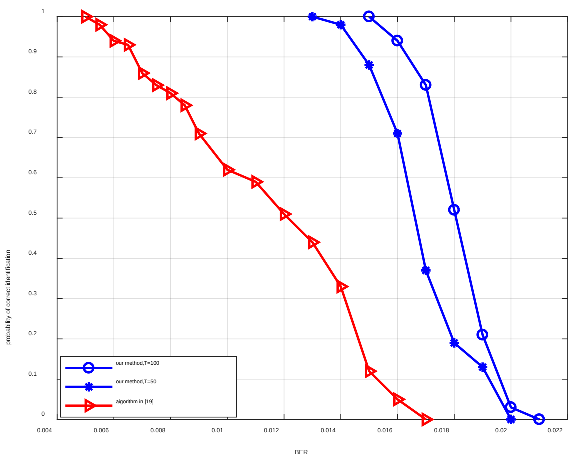

Fig. 5. The probability of identification of $(8Z,4Z)$ LDPC code with respect to different SNR from the algorithm in [19] and our method for $T=50$ and $T=100$, respectively.

Table 2. The locations of nonzero elements of the derived parity-checks

1	21,40,84,100	10	6,54,71,86
2	9,43,71,118	11	2,36,80,127
3	3,51,68,83	12	28,64,74,106,122
4	20,39,83,99	13	39,53,67,70,85,114

5	28,47,91,107	14	9,26,45,57,74,105
6	15,33,77,124	15	7,19,65,72,87,97,113
7	7,41,69,116	16	10,14,58,62,75,79,90,94
8	2,50,67,82	17	15,29,48,50,76,77,92,93,109
9	10,58,75,90	18	7,28,41,64,69,74,106,116,122

Table 3. The weight of $w(\boldsymbol{A}_c h')$ for different Z

$Z = 4$	
$\boldsymbol{h}(21, 40, 84, 100)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(21, 40, 84, 100)) = 53$
$\boldsymbol{h}(22, 37, 81, 97)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(22, 37, 81, 97)) = 631$
$\boldsymbol{h}(23, 38, 82, 98)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(23, 38, 82, 98)) = 600$
$\boldsymbol{h}(24, 39, 83, 99)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(24, 39, 83, 99)) = 667$
$Z = 8$	
$\boldsymbol{h}(21, 40, 84, 100)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(21, 40, 84, 100)) = 53$
$\boldsymbol{h}(22, 33, 85, 101)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(22, 33, 85, 101)) = 636$
$\boldsymbol{h}(23, 34, 86, 102)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(23, 34, 86, 102)) = 642$
$\boldsymbol{h}(24, 35, 87, 103)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(24, 35, 87, 103)) = 627$
$\boldsymbol{h}(17, 36, 88, 104)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(17, 36, 88, 104)) = 602$
$\boldsymbol{h}(18, 37, 81, 97)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(18, 37, 81, 97)) = 70$
$\boldsymbol{h}(19, 38, 82, 98)$	$w(\boldsymbol{A}_c \cdot \boldsymbol{h}'(19, 38, 82, 98)) = 65$

$\boldsymbol{h}(20,\ 39,\ 83,\ 99)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(20,\ 39,\ 83,\ 99)\right) = 62$
$Z = 16$	
$\boldsymbol{h}(21,\ 40,\ 84,\ 100)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(21,\ 40,\ 84,\ 100)\right) = 53$
$\boldsymbol{h}(22,\ 41,\ 85,\ 101)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(22,\ 41,\ 85,\ 101)\right) = 57$
$\boldsymbol{h}(23,\ 42,\ 86,\ 102)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(23,\ 42,\ 86,\ 102)\right) = 64$
$\boldsymbol{h}(24,\ 43,\ 87,\ 103)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(24,\ 43,\ 87,\ 103)\right) = 61$
$\boldsymbol{h}(25,\ 44,\ 88,\ 104)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(25,\ 44,\ 88,\ 104)\right) = 65$
$\boldsymbol{h}(26,\ 45,\ 89,\ 105)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(26,\ 45,\ 89,\ 105)\right) = 60$
$\boldsymbol{h}(27,\ 46,\ 90,\ 106)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(27,\ 46,\ 90,\ 106)\right) = 46$
$\boldsymbol{h}(28,\ 47,\ 91,\ 107)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(28,\ 47,\ 91,\ 107)\right) = 65$
$\boldsymbol{h}(29,\ 48,\ 92,\ 108)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(29,\ 48,\ 92,\ 108)\right) = 52$
$\boldsymbol{h}(30,\ 33,\ 93,\ 109)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(30,\ 33,\ 93,\ 109)\right) = 60$
$\boldsymbol{h}(31,\ 34,\ 94,\ 110)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(31,\ 34,\ 94,\ 110)\right) = 58$
$\boldsymbol{h}(32,\ 35,\ 95,\ 111)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(32,\ 35,\ 95,\ 111)\right) = 56$
$\boldsymbol{h}(17,\ 36,\ 96,\ 112)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(17,\ 36,\ 96,\ 112)\right) = 61$
$\boldsymbol{h}(18,\ 37,\ 81,\ 97)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(18,\ 37,\ 81,\ 97)\right) = 70$
$\boldsymbol{h}(19,\ 38,\ 82,\ 98)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(19,\ 38,\ 82,\ 98)\right) = 65$
$\boldsymbol{h}(20,\ 39,\ 83,\ 99)$	$w\left(\boldsymbol{A}_c \cdot \boldsymbol{h}'(20,\ 39,\ 83,\ 99)\right) = 62$

5. Conclusion

In this paper, a novel two-step screening method is proposed to deal with the problem of blind identification of block codes without a candidate set. In the first step of screening, we resolve some vectors which satisfy most of the given linear error equations and put them in a set Θ. In the second step, we choose the members which make $w\left(\boldsymbol{A}_{check} \cdot \boldsymbol{h}^{\mathrm{T}}\right) < \beta$ from Θ using a check matrix $\boldsymbol{A}_{check}$. Then we can reconstruct the parity-check matrix as long as we can derive enough linear independent vectors. Simulation results demonstrate that our method has a better fault-tolerant ability than those using GCE as the main step. In the future work, we plan to combine the advantages of our method with that in [19] to solve the identification problem of linear block codes of a large block length without a quasi-cyclic structure.

References

[1] R. G. Gallager, *Information theory and reliable communication*, New York: Wiley, 1968.

[2] A. Goldsmith, S. G. Chua, “Adaptive coded modulation for fading channels,” *IEEE Trans. Commun.*, vol. 46, no. 5, pp. 595–602, May 1998. Article(CrossRefLink)

[3] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Trans. Inform. Theory*, vol. 43, no. 6, pp. 1757–1766, 1997. Article(CrossRefLink)
[4] R. Moosavi, E. G. Larsson, "Fast blind recognition of channel codes," *IEEE Trans. on Comm.*, vol. 62, no. 5, pp.1393–1405, May 2014. Article(CrossRefLink)
[5] A.Valembois, "Detection and recognition of a binary linear code," *Discrete Applied Mathematics*, vol. 111, pp. 199–218, 2001. Article(CrossRefLink)
[6] C. Chabot, "Recognition of a code in a noisy environment," in *Proc. of ISIT07*, Nice, France, pp. 2210–2215, 2007. Article(CrossRefLink)
[7] X. Tian, H. WU, "Novel blind identification LDPC codes using average LLR of syndrome a posteriori probability," *IEEE Transactions on Signal Processing*, vol. 62, no. 3, pp. 632–640, 2014. Article(CrossRefLink)
[8] P. Yu, H. Peng, and J. Li, "On blind recognition of channel codes within a candidate set," *IEEE Commun. Lett.*, vol. 20, no. 4, pp. 736–739, 2016. Article(CrossRefLink)
[9] Z. Wu, et al., "Blind Recognition of LDPC Codes over Candidate Set," *IEEE Commun. Lett.*, vol. 24, no. 1, pp. 11–14, 2020. Article(CrossRefLink)
[10] X. Lv, Z. Huang, and S. Su, "Fast recognition method for generator polynomial of BCH codes," *J. Xidian. Univ.*, vol. 38, no. 6, pp. 159-162, 2011. Article(CrossRefLink)
[11] R. Imad, C. Poulliat, S. Houcke, and G. Gadat, "Blind frame synchronization of Reed-Solomon codes: non-binary vs. binary approach," in *Proc. of IEEE SPAWC 2010*, Marrakech, Morocco, 2010. Article (CrossRef Link)
[12] J. Jiang, K. R. Narayanan, "Iterative soft-input-soft-output decoding of Reed-Solomon codes by adapting the parity check matrix," *IEEE. Trans. Infor. Theory*, vol. 52, no. 8, pp. 3746–3756, 2006. Article(CrossRefLink)
[13] M. Marazin, R. Gautier, and G. Burel, "Blind recovery of k/n rate convolutional encoders in a noisy environment," *EURASIP J. Wirel. Commun. Netw*, vol. 168, pp.1–9, 2011. Article(CrossRefLink)
[14] Y. G. Debessu, H. C. Wu, and J. Hong, "Novel blind encoder parameter estimation for turbo codes," *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 1917–1920, 2012. Article (CrossRef Link)
[15] P. Yu, J. Li, H. Peng, "A least square method for parameter estimation of RSC sub-codes of turbo codes," *IEEE Commun. Lett.*, vol. 18, no. 4, pp. 644–647, 2014. Article(CrossRefLink)
[16] M. Cluzeau, "Block code reconstruction using iterative decoding techniques," in *Proc. of ISIT06*, Seattle, USA, pp. 2269–2273, 2006. Article(CrossRefLink)
[17] J. Barbier, G. Sicot, and S. Houcke, "Algebraic approach for the reconstruction of linear and convolutional error correcting codes," in *Proc. of World Academy of Science Engineering & Technology*, vol. 2, no. 3, pp. 113–118, 2006.
[18] G. Sicot, S. Houcke, and J. Barbier, "Blind detection of interleaver parameters," *Signal Process.*, vol. 89, pp. 450–462, 2009. Article(CrossRefLink)
[19] W. Wang, H. Peng, and J. Li, "Blind Identification of LDPC Codes Based on Decoding," in *Proc. of 2017 ICCTEC*, Dalian, China, pp. 998–1001, 2017. Article(CrossRefLink)
[20] Q. Liu, H. Zhang, and G. Shen, et al, "A Fast Reconstruction of the Parity-Check Matrices of LDPC Codes in a Noisy Environment," *Computer Communications*, vol. 176, pp.163–172, 2021. Article (CrossRef Link)
[21] J. Xia, "Linear error equation on field F_2," *Chin. Quart. J. of Math*, vol. 22, no. 4, pp. 518–522, 2007.
[22] R. Imad, G. Sicot and S. Houcke, "Blind frame synchronization for error correcting codes having a sparse parity check matrix," *IEEE Transactions on Communications*, vol. 57, no. 6, pp. 1574–1577, June 2009. Article(CrossRefLink)
[23] R. Imad, S. Houcke, "Theoretical analysis of a MAP based blind frame synchronizer," *IEEE Transactions on Wireless Communications*, vol. 8, no. 11, pp. 5472–5476, Nov. 2009. Article(CrossRefLink)
[24] M. Cluzeau, M. Finiasz, "Recovering a code's length and synchronization from a noisy intercepted bitstream," in *Proc. of ISIT2009,* Seoul, Korea, June28–July3, 2009. Article(CrossRef Link)

[25] Y. Zrelli, M. Marazin, R. Gautier, et al., "Blind identification of code word length for non-binary error-correcting codes in noisy transmission," *EURASIP J. Wirel. Commun. Netw.*, vol. 43, pp. 1–16, 2015. Article(CrossRefLink)

[26] S. Ramabadran, A. S. Madhu Kumar, et al., "Blind recognition of LDPC code parameters over erroneous channel conditions," *IET Signal Process.*, vol. 13, no. 1, pp. 86–95, 2019. Article(CrossRefLink)

[27] Z. P. Shi, *Advanced channel coding for 5G communication systems*, Peking, China: post & telecom press, 2017, pp. 91.

[28] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, IT-8, vol. 8, no. 1, pp. 21–28, 1962. Article(CrossRefLink)

Qian Liu was born in Cangzhou, Hebei province, China in 1984. She majored in basic mathematics and received the B. S. and M. S. degrees from Zhengzhou University, Zhengzhou, China, in 2007 and 2010, respectively. She has been working in Information Engineering University up to now. Currently she is working towards a Ph. D. degree at Information Engineering University. Her major research interests are in the areas of signal processing.

Hao Zhang was born in Zhengzhou, Henan province, China in 1984. He received his M. S. degree in applied mathematics and Ph. D. in signal and information processing from Information Engineering University, respectively. He has been working in Information Engineering University up to now. His research interests include Fourier analysis, information hiding and error coding.

Peidong Yu was born in Cili, Hunan province, China in 1988. He received his B. S. degree from Tsinghua University, received his M. S. degree and Ph. D. in identification of parameters of channel coding from Information Engineering University, respectively. Now he is a lecturer in Information Engineering University.

Gang Wang was born in Chuzhou, Anhui province, China in 1980. He majored in signal processing and received his B. S. degree, M. S. degree and Ph. D. degree from Information Engineering University in 2003, 2006 and 2019, respectively. He has been working in Information Engineering University up to now.

Zhaoyang Qiu was born in Xuchang, Henan, China, in 1991. He received his M. S. degree from Information Engineering University (IEU), Zhengzhou, Henan, in 2017, where he is currently pursuing a Ph.D. degree in information and communication engineering. His main research interests include wireless communication and signal processing, and information security.