

A Random Coding Approach to Performance Analysis of the Ordered Statistic Decoding With Local Constraints

Jifan Liang and Xiao Ma

Abstract

This paper is concerned with the ordered statistic decoding with local constraints (LC-OSD) of binary linear block codes, which is a near maximum-likelihood decoding algorithm. Compared with the conventional OSD, the LC-OSD significantly reduces both the maximum and the average number of searches. The former is achieved by performing the serial list Viterbi algorithm (SLVA) or a two-way flipping pattern tree (FPT) algorithm with local constraints on the test error patterns, while the latter is achieved by incorporating tailored early termination criteria. The main objective of this paper is to explore the relationship between the performance of the LC-OSD and decoding parameters, such as the constraint degree and the maximum list size. To this end, we approximate the local parity-check matrix as a totally random matrix and then estimate the performance of the LC-OSD by analyzing with a saddlepoint approach the performance of random codes over the channels associated with the most reliable bits (MRBs). The random coding approach enables us to derive an upper bound on the

This work was supported in part by the National Key R&D Program of China (No. 2021YFA1000500). (*Corresponding author: Xiao Ma.*)

The authors are with the School of Computer Science and Engineering and the Guangdong Key Laboratory of Information Security Technology, Sun Yat-sen University, Guangzhou 510006, China (e-mail: liangjf56@mail2.sysu.edu.cn; maxiao@mail.sysu.edu.cn).

performance and predict the average rank of the transmitted codeword in the list delivered by the LC-OSD. This allows us to balance the constraint degree and the maximum list size for the average (or maximum) time complexity reduction. Simulation results show that the approximation by random coding approach is numerically effective and powerful. Simulation results also show that the RS codes decoded by the LC-OSD can approach the random coding union (RCU) bounds, verifying the efficiency and universality of the LC-OSD.

Index Terms

List size, ordered statistic decoding with local constraints (LC-OSD), performance bounds, random codes, serial list Viterbi algorithm (SLVA).

I. INTRODUCTION

As a central problem in coding theory, the maximum-likelihood decoding (MLD) of a general binary linear code is considered to have no efficient algorithms due to its NP-completeness [1]. The ordered statistics decoding (OSD) investigated by Fossorier and Lin [2] is a near-optimal list decoding algorithm but requires a large number of searches, leading to high complexity. The OSD algorithm first sorts the reliability vector and then selects k linearly independent positions with the most reliabilities, referred to as the most reliable basis. By re-encoding, the OSD lists all codewords with Hamming distance no greater than t (a preset nonnegative integer) in the MRB and delivers the most likely candidate codeword as output. As a list decoding algorithm, three issues about OSD naturally arise: 1) for what candidate codewords to search, 2) in which order to search, and 3) when to stop the search. To solve these issues, many improvements to OSD have been proposed. For reducing search space, segmentation-discarding OSD (SD-OSD) [3] and linear-equation OSD (LE-OSD) [4] have been proposed; for early stopping, probability-based OSD (PB-OSD) [5] can significantly reduce the number of searches.

Recently, a variant of OSD called OSD with local constraints (LC-OSD) was proposed further to reduce the decoding complexity [6], [7]. The basic idea of the LC-OSD is to consider an

extended subset of reliable positions by including δ extra positions next to the MRB, which was proposed in a report [8] of NASA as early as 1979 and widely used in the information set decoding (ISD) algorithms of cryptography [9]–[11]. The local constraints can be characterized as a trellis specified by a local parity-check matrix, and a list of candidate codewords can be generated by performing the serial list Viterbi algorithm (SLVA) [6]. Aided by tailored early stopping criteria, the LC-OSD can reduce the number of [searches/re-encodings](#) to within ten in the high signal-to-noise ratio (SNR) region [7]. We further proposed an algorithm, the two-way flipping pattern tree, for replacing the SLVA in the LC-OSD algorithm to reduce the overhead of the SLVA in the high SNR (low average number of searches) region.

The main contributions of the manuscript are listed below.

- 1) Theoretically, we derived an upper bound on the performance gap between the LC-OSD and the MLD performance for a general code. The performance gap is estimated by the random coding approach. This implies that the gap between the LC-OSD and the MLD is less relevant to the code structure, suggesting the universality of the LC-OSD. In particular, simulation results show that RS codes can approach the random coding union (RCU) bound in a wide range of code rates. We also proposed a semi-simulation method to speed up the calculation of the proposed bound, which is powered by the saddlepoint method [12], [13].
- 2) Based on the idea of “search-on-demand,” we provided in Sec. IV-A an implementation of SLVA with an intuitive example. For a $\mathcal{C}[N, K]$ binary linear codes, the time complexity of this implementation is $\mathcal{O}(N(2^{N-K} + \ell))$, which is highly reduced from the PLVA (parallel list Viterbi algorithm), where ℓ denotes the average number of searches. This implementation is asymptotically lower than that of [14] and [15] with complexities of $\mathcal{O}(N(2^{N-K} + \ell^2))$ and $\mathcal{O}(N(2^{N-K} + \ell) + \ell^2)$, respectively.
- 3) The list-generating algorithm (LGA) is a procedure generating a list of codewords, which is used in searching for MRB codewords (Sec. III-B2). As a substitution, we provide an

algorithm, the two-way flipping pattern tree (tFPT) algorithm in Sec. IV-B, to reduce the overhead of producing the first candidate. The tFPT algorithm uses a similar technology as the box-and-match (BMA) algorithm [16], which reduces computation in the high SNR region. More details about tFPT algorithm are given in Sec. IV-B.

The rest of this paper is organized as follows. In Sec. II, we illustrate the system model, describe the implementation of the LC-OSD, and analyze the complexity of the LC-OSD. In Sec. III, we derive the theoretical results and present some related examples. In Sec. IV, we present two LGAs, the SLVA and the tFPT and their performance and/or complexity (throughput).

II. LC-OSD

A. System Model

Let $\mathbb{F}_2 = \{0, 1\}$ be the binary field¹ and $\mathcal{C}[n, k]$ be a binary linear code of length n and dimension k . Let \mathbf{G} be a generator matrix of \mathcal{C} and \mathbf{H} be a parity-check matrix of \mathcal{C} . The generator matrix of size $k \times n$ and the parity-check matrix of size $(n - k) \times n$ are related by

$$\mathbf{G}\mathbf{H}^T = \mathbf{O}. \quad (1)$$

Let $\mathbf{u} \in \mathbb{F}_2^k$ be an information vector to be transmitted, which is first encoded into $\mathbf{c} \in \mathbb{F}_2^n$ by

$$\mathbf{c} = \mathbf{u}\mathbf{G}, \quad (2)$$

and then modulated by the binary phase shift keying (BPSK) into a signal vector $\mathbf{x} \in \mathbb{R}^n$ as

$$x_i = (-1)^{c_i}, \quad 1 \leq i \leq n. \quad (3)$$

Then the signal vector \mathbf{x} is transmitted over an AWGN channel, resulting in a received vector $\mathbf{y} \in \mathbb{R}^n$ given by

$$\mathbf{y} = \mathbf{x} + \mathbf{w}, \quad (4)$$

¹In this paper, when it is clear from the context, we also convert naturally 0 and 1 in \mathbb{F}_2 to the real numbers.

where $\mathbf{w} \sim \text{Normal}(\mathbf{0}, \sigma^2 \mathbf{I}_n)$ is a sample vector of white Gaussian noise. The bit-wise hard-decision vector $\mathbf{z} \in \mathbb{F}_2^n$ is given by

$$z_i \triangleq \begin{cases} 0, & \text{if } y_i \geq 0 \\ 1, & \text{if } y_i < 0 \end{cases}, \quad 1 \leq i \leq n. \quad (5)$$

The log-likelihood ratio (LLR) vector, denoted by $\mathbf{r} \in \mathbb{R}^n$, is defined as

$$r_i \triangleq \log \frac{f_{y|c}(y_i | 0)}{f_{y|c}(y_i | 1)} = \frac{2y_i}{\sigma^2}, \quad 1 \leq i \leq n, \quad (6)$$

where $f_{y|c}(\cdot | \cdot)$ is the conditional probability density function. The reliability of z_i refers to as $|r_i|$. Also, the test error pattern (TEP) $\mathbf{e} \in \mathbb{F}_2^n$ for a test codeword $\mathbf{v} \in \mathbb{F}_2^n$ is given by

$$\mathbf{e} \triangleq \mathbf{z} - \mathbf{v}. \quad (7)$$

The MLD is to find one of the codewords \mathbf{v}^* such that²

$$\mathbf{v}^* = \arg \max_{\mathbf{v} \in \mathcal{C}} f_{\mathbf{y}|c}(\mathbf{y} | \mathbf{v}), \quad (8)$$

which is equivalent to

$$\mathbf{v}^* = \arg \min_{\mathbf{v} \in \mathcal{C}} \log \frac{f_{\mathbf{y}|c}(\mathbf{y} | \mathbf{z})}{f_{\mathbf{y}|c}(\mathbf{y} | \mathbf{v})}. \quad (9)$$

Definition 1. (Soft weight³) The soft weight of the TEP $\mathbf{e} \in \mathbb{F}_2^n$ is defined as

$$\Gamma(\mathbf{e}) \triangleq \log \frac{f_{\mathbf{y}|c}(\mathbf{y} | \mathbf{z})}{f_{\mathbf{y}|c}(\mathbf{y} | \mathbf{z} - \mathbf{e})} = \sum_{i=1}^n \log \frac{f_{y|c}(y_i | z_i)}{f_{y|c}(y_i | z_i - e_i)} = \sum_{i=1}^n e_i |r_i|. \quad (10)$$

For convenience, we also write the soft weight of \mathbf{e} as an inner product. That is,

$$\Gamma(\mathbf{e}) = \langle \mathbf{e}, |\mathbf{r}| \rangle \quad (11)$$

where $|\cdot|$ denotes the element-wise absolute value.

We see that the MLD is equivalent to the lightest-soft-weight decoding.

²If two or more codewords achieve the maximum, we simply select one at random.

³This is called weighted Hamming distance in [17] and ellipsoidal weight in [18].

B. OSD With Local Constraints

In this subsection, we focus on an arbitrary binary linear code $\mathcal{C}[n, k]$ specified by a parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$. For a preset constraint degree δ ($0 \leq \delta \leq n - k$) and a maximum number of searches ℓ_{\max} , we restate the LC-OSD algorithm in the following.

- 1) Upon receiving \mathbf{y} , calculate the bit-wise hard-decision vector \mathbf{z} by (5) and the LLR vector \mathbf{r} by (6).
- 2) Sort the LLR vector \mathbf{r} into $\mathbf{r}\mathbf{\Pi}$ such that the first $(n - k - \delta)$ **columns of the corresponding permuted parity-check matrix, $\mathbf{H}\mathbf{\Pi}$** , are linearly independent with least reliabilities. To be precise, $\mathbf{\Pi}$ is a permutation matrix satisfying:
 - a) the first $(n - k - \delta)$ columns of $\mathbf{H}\mathbf{\Pi}$ is column full rank, i.e., $\text{rank}(\mathbf{H}\mathbf{\Pi})_{[1, n-k-\delta]} = n - k - \delta$, and
 - b) $\sum_{i=1}^{n-k-\delta} (|\mathbf{r}|\mathbf{\Pi})_i$ is as small as possible, where $(|\mathbf{r}|\mathbf{\Pi})_i$ stands for the i -th element of the permuted reliability vector $(|\mathbf{r}|\mathbf{\Pi})$.

This method is equivalent to that of the box-and-match (BMA) [16] algorithm and can be accomplished by a greedy algorithm, summarized in Algorithm 1 for completeness.

Algorithm 1 Find a permutation matrix for [the LC-OSD](#)

Input: The parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}$, the reliability vector $|\mathbf{r}| \in \mathbb{R}^n$, the constraint degree δ ($0 \leq \delta \leq n - k$).

Output: A permutation matrix $\mathbf{\Pi}$ satisfying the requirements (2a–2b).

```

1: function GET-PERMUTATION( $\mathbf{H}, |\mathbf{r}|, \delta$ )
2:    $\mathbf{\Pi} \leftarrow$  an ascending sorted permutation of vector  $|\mathbf{r}|$   $\triangleright$  That is,  $(|\mathbf{r}|\mathbf{\Pi})$  is non-decreasing.
3:    $i' \leftarrow n - k - \delta + 1$ 
4:   for  $i = 1, 2, \dots, n - k - \delta$  do
5:     while  $\text{rank}(\mathbf{H}\mathbf{\Pi})_{[1,i]} < i$  do  $\triangleright (\mathbf{H}\mathbf{\Pi})_{[1,i]}$  is formed by the first  $i$  columns of  $\mathbf{H}\mathbf{\Pi}$ .
6:        $\text{swap}(\mathbf{\Pi}_i, \mathbf{\Pi}_{i'})$   $\triangleright$  Swap the columns  $i$  and  $i'$  of  $\mathbf{\Pi}$ .
7:        $i' \leftarrow i' + 1$ 
8:   return  $\mathbf{\Pi}$ 

```

For convenience, we denote the permuted \mathbf{z} by $\tilde{\mathbf{z}}$, i.e.,

$$\tilde{\mathbf{z}} \triangleq \mathbf{z}\mathbf{\Pi}, \quad (12)$$

and use similar notations for vectors $\mathbf{r}, \mathbf{e}, \mathbf{v}$ etc.

3) Perform the Gaussian elimination for $\mathbf{H}\mathbf{\Pi}$, resulting in $\tilde{\mathbf{H}} = \mathbf{Q}\mathbf{H}\mathbf{\Pi}$ of form

$$\tilde{\mathbf{H}} = \left[\begin{array}{c|c} \begin{matrix} n-k-\delta \text{ columns} & k+\delta \text{ columns} \end{matrix} \\ \hline \begin{matrix} \mathbf{I} & \mathbf{P}_1 \end{matrix} \\ \hline \begin{matrix} \mathbf{O} & \mathbf{P}_2 \end{matrix} \end{array} \right] \begin{matrix} n-k-\delta \text{ rows} \\ \delta \text{ rows} \end{matrix} \quad (13)$$

where \mathbf{I} denotes the identity matrix of order $n - k - \delta$.

4) A test vector $\mathbf{v} = \mathbf{z} - \mathbf{e}$ is a codeword of \mathcal{C} if and only if the TEP \mathbf{e} satisfies the

parity-check equation

$$\mathbf{v}\mathbf{H}^T = (\mathbf{z} - \mathbf{e})\mathbf{H}^T = \mathbf{0}, \quad (14)$$

or equivalently, in the permuted form,

$$\tilde{\mathbf{v}}\tilde{\mathbf{H}}^T = (\tilde{\mathbf{z}} - \tilde{\mathbf{e}})\tilde{\mathbf{H}}^T = \mathbf{0}. \quad (15)$$

So, searching a (permuted) codeword $\tilde{\mathbf{v}}$ is equivalent to searching a TEP $\tilde{\mathbf{e}}$ because $\tilde{\mathbf{z}}$ is fixed upon receiving \mathbf{y} . If we divide $\tilde{\mathbf{e}}$ (and $\tilde{\mathbf{z}}$) into two parts of lengths $(n - k - \delta)$ and $(k + \delta)$, namely

$$\tilde{\mathbf{e}} = \left[\begin{array}{c|c} \overset{n-k-\delta \text{ bits}}{\tilde{\mathbf{e}}_L} & \overset{k+\delta \text{ bits}}{\tilde{\mathbf{e}}_R} \end{array} \right], \quad \tilde{\mathbf{z}} = \left[\begin{array}{c|c} \overset{n-k-\delta \text{ bits}}{\tilde{\mathbf{z}}_L} & \overset{k+\delta \text{ bits}}{\tilde{\mathbf{z}}_R} \end{array} \right], \quad (16)$$

where the left part is known as the least reliable bits (LRBs) and the right part is the MRBs. Then by expanding (15), we can obtain equations with unknowns $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_L, \tilde{\mathbf{e}}_R)$,

$$\tilde{\mathbf{e}}_L + \tilde{\mathbf{e}}_R \mathbf{P}_1^T = \tilde{\mathbf{s}}_1, \quad (17)$$

$$\tilde{\mathbf{e}}_R \mathbf{P}_2^T = \tilde{\mathbf{s}}_2, \quad (18)$$

where the right-hand sides (RHSs) of (17) and (18) are given by

$$\tilde{\mathbf{s}}_1 \triangleq \tilde{\mathbf{z}}_L + \tilde{\mathbf{z}}_R \mathbf{P}_1^T, \quad (19)$$

$$\tilde{\mathbf{s}}_2 \triangleq \tilde{\mathbf{z}}_R \mathbf{P}_2^T, \quad (20)$$

which will be calculated and stored before searching codewords to avoid replicated computations.

As in (10), the soft weight of $\tilde{\mathbf{e}}$ is denoted as⁴

$$\Gamma(\tilde{\mathbf{e}}) \triangleq \langle \tilde{\mathbf{e}}, |\tilde{\mathbf{r}}| \rangle. \quad (21)$$

⁴Strictly speaking, it should use $\tilde{\Gamma}(\tilde{\mathbf{e}})$ instead of $\Gamma(\tilde{\mathbf{e}})$ for $\tilde{\mathbf{r}}$.

Similarly,

$$\Gamma(\tilde{\mathbf{e}}_{\text{L}}) \triangleq \langle \tilde{\mathbf{e}}_{\text{L}}, |\tilde{\mathbf{r}}_{\text{L}}| \rangle, \Gamma(\tilde{\mathbf{e}}_{\text{R}}) \triangleq \langle \tilde{\mathbf{e}}_{\text{R}}, |\tilde{\mathbf{r}}_{\text{R}}| \rangle. \quad (22)$$

We have

$$\Gamma(\tilde{\mathbf{e}}) = \Gamma(\tilde{\mathbf{e}}_{\text{L}}) + \Gamma(\tilde{\mathbf{e}}_{\text{R}}). \quad (23)$$

- 5) From (17), we see that $\tilde{\mathbf{e}}_{\text{L}}$ (and $\tilde{\mathbf{e}}$ by (16)) are uniquely determined by $\tilde{\mathbf{e}}_{\text{R}}$. Hence, we only need to search for $\tilde{\mathbf{e}}_{\text{R}}$ instead of $\tilde{\mathbf{e}}$ (or $\tilde{\mathbf{v}}$). Intuitively, we should make a list of $\tilde{\mathbf{e}}_{\text{R}}$

$$\tilde{\mathbf{e}}_{\text{R}}^{(1)}, \tilde{\mathbf{e}}_{\text{R}}^{(2)}, \dots, \tilde{\mathbf{e}}_{\text{R}}^{(\ell_{\text{max}})} \quad (24)$$

satisfying (18) in an order such that the partial soft weights are non-decreasing, i.e.,

$$\Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(1)}) \leq \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(2)}) \leq \dots \leq \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(\ell_{\text{max}})}). \quad (25)$$

This can be achieved by an LGA, such as the SLVA [14] and the tFPT. See Sec. IV for details.

- 6) For each $\tilde{\mathbf{e}}_{\text{R}}^{(j)}$ ($1 \leq j \leq \ell_{\text{max}}$), solve for $\tilde{\mathbf{e}}^{(j)}$ by (17) and (16).
 7) Finally, output the codeword corresponding to one of the $\tilde{\mathbf{e}}^{(j)}$ with the lightest soft weight.

For the sake of clarity, the LC-OSD is summarized in Algorithm 2 with an optional early stopping criterion (see Sec. III-A for details).

Algorithm 2 OSD with local constraints (LC-OSD)**Input:** \mathbf{H} , \mathbf{y} , δ , ℓ_{\max} , an early stopping criterion Ψ (optional).**Output:** The optimal searched codeword \mathbf{v} .

```

1: function LC-OSD( $\mathbf{H}$ ,  $\mathbf{y}$ ,  $\delta$ ,  $\ell_{\max}$ ,  $\Psi$ )
2:    $\mathbf{z} \leftarrow$  the bit-wise hard decision vector of  $\mathbf{y}$   $\triangleright$  Calculate by (5).
3:    $\mathbf{r} \leftarrow$  the LLR vector of  $\mathbf{y}$   $\triangleright$  Calculate by (6).
4:    $\mathbf{\Pi} \leftarrow$  GET-PERMUTATION( $\mathbf{H}$ ,  $|\mathbf{r}|$ ,  $\delta$ )  $\triangleright$  Get a permutation matrix by Algorithm 1.
5:   Permute  $\mathbf{z}$ ,  $\mathbf{r}$  into  $\tilde{\mathbf{z}}$ ,  $\tilde{\mathbf{r}}$  by  $\mathbf{\Pi}$   $\triangleright$  Permute as (12).
6:    $\tilde{\mathbf{H}} \leftarrow$  Gaussian eliminated matrix of  $\mathbf{H}\mathbf{\Pi}$   $\triangleright$  Eliminate into the form as (13).
7:    $\mathbf{P}_1, \mathbf{P}_2 \leftarrow$  the top-right and bottom-right sub-matrices of  $\tilde{\mathbf{H}}$   $\triangleright$  Extract as (13).
8:    $\tilde{\mathbf{s}}_1 \leftarrow \tilde{\mathbf{z}}_L + \tilde{\mathbf{z}}_R \mathbf{P}_1^T$   $\triangleright$  Calculate the RHS of (19).
9:    $\tilde{\mathbf{s}}_2 \leftarrow \tilde{\mathbf{z}}_R \mathbf{P}_2^T$   $\triangleright$  Calculate the RHS of (20).
10:   $\tilde{\mathbf{e}}_{\text{opt}} \leftarrow \tilde{\mathbf{z}}$   $\triangleright$  Note that  $\tilde{\mathbf{z}}$  is a valid TEP for test.
11:  for  $\ell = 1, 2, \dots, \ell_{\max}$  do
12:     $\tilde{\mathbf{e}}_R^{(\ell)} \leftarrow$  LGA( $\mathbf{P}_2$ ,  $|\tilde{\mathbf{r}}_R|$ ,  $\tilde{\mathbf{s}}_2$ ,  $\ell$ )  $\triangleright$  Generate the  $\ell$ -th lightest TEP (Sec. IV).
13:     $\tilde{\mathbf{e}}_L^{(\ell)} \leftarrow \tilde{\mathbf{s}}_1 - \tilde{\mathbf{e}}_R^{(\ell)} \mathbf{P}_1^T$   $\triangleright$  Solve from (17).
14:     $\tilde{\mathbf{e}}^{(\ell)} \leftarrow (\tilde{\mathbf{e}}_L^{(\ell)}, \tilde{\mathbf{e}}_R^{(\ell)})$ 
15:    if  $\Gamma(\tilde{\mathbf{e}}_{\text{opt}}) > \Gamma(\tilde{\mathbf{e}}^{(\ell)})$  then  $\triangleright$  Update the optimal TEP with LSW.
16:       $\tilde{\mathbf{e}}_{\text{opt}} \leftarrow \tilde{\mathbf{e}}^{(\ell)}$ 
17:    if the early stopping criterion  $\Psi$  is satisfied then  $\triangleright$  See Sec.III-A for details.
18:      break
19:   $\mathbf{v} \leftarrow \mathbf{z} - \tilde{\mathbf{e}}_{\text{opt}} \mathbf{\Pi}^{-1}$ 
20:  return  $\mathbf{v}$ 

```

Remarks. If $\delta = 0$, Algorithm 2 reduces to the OSD algorithm [2] with a search order the same as that of the FPT; if $\delta = n - k$, it is indeed the MLD.

C. Complexity Analysis

In this subsection, we focus on the time and space complexity of the **LC-OSD (Algorithm 2)**.

1) *Time Complexity*: We analyze the time complexity of the algorithm by evaluating the number of floating point operations (FLOPs) or binary operations (BOPs) required by each step. The time complexity consists of three dominant parts:

- 1) *Gaussian elimination*. Reducing \mathbf{H} into a block upper triangular matrix as shown in (13) requires $\mathcal{O}((n-k)(n-k-\delta))$ elementary row operations, each taking $\mathcal{O}(n)$ BOPs.
- 2) *Re-encoding*. With initializations done, searching a candidate $\tilde{\mathbf{e}}_{\mathbf{R}}^{(j)}$ ($j > 1$) by the SLVA (line 12, Algorithm 2) requires $\mathcal{O}(n)$ FLOPs and re-encoding $\tilde{\mathbf{e}}_{\mathbf{L}}^{(j)}$ by (17) (line 13, Algorithm 2) requires $\mathcal{O}((n-k-\delta)(k+\delta))$ BOPs (recalling that \mathbf{P}_1 is of size $(n-k-\delta) \times (k+\delta)$).
- 3) *The LGA*. As analyzed in Sec. IV, different LGAs may have different time complexities, denoted by T^{LGA} .

To summarize, the overall average time complexity is

$$T_{\text{avg}}^{\text{tot}} = \underbrace{\mathcal{O}(n(n-k)(n-k-\delta))}_{\text{Gaussian elimination}} + \underbrace{\ell_{\text{avg}} \mathcal{O}((n-k-\delta)(k+\delta))}_{\text{re-encoding}} + \underbrace{T_{\text{avg}}^{\text{LGA}}}_{\text{the LGA}}, \quad (26)$$

where ℓ_{avg} denotes the average number⁵ of searches per frame, and the overall worst-case time complexity is

$$T_{\text{max}}^{\text{tot}} = \mathcal{O}(n(n-k)(n-k-\delta)) + \ell_{\text{max}} \mathcal{O}((n-k-\delta)(k+\delta)) + T_{\text{max}}^{\text{LGA}}. \quad (27)$$

2) *Space Complexity*: We analyze the space complexity of the algorithm by calculating the number of bytes used. The space complexity consists of two dominant parts:

- 1) *Matrices and vectors*. The algorithm needs to store matrices \mathbf{H} , \mathbf{P}_1 and \mathbf{P}_2 of size $\mathcal{O}((n-k) \times n)$ and several vectors of length $\mathcal{O}(n)$.
- 2) *The LGA*. Different LGAs may have different space complexities, denoted by S^{LGA} .

⁵Usually, ℓ_{avg} is much less than ℓ_{max} due to the early stopping criteria (Sec. III-A).

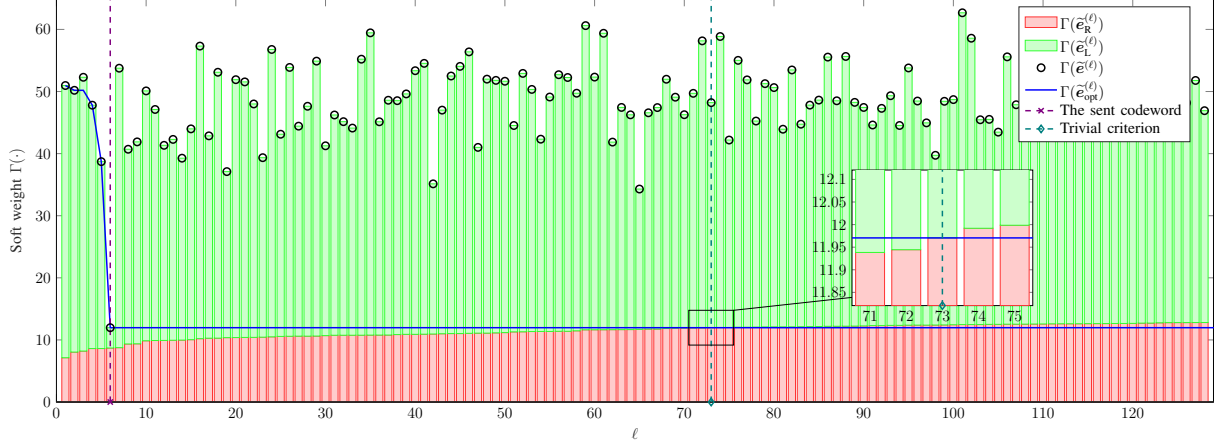


Fig. 1. Numerical illustration of the relationships among soft weights $\Gamma(\cdot)$. The data are obtained by simulating a $\mathcal{C}[128, 64]$ code at $E_b/N_0 = 2.5$ dB, where the LC-OSD parameters are $(\delta, \ell_{\max}) = (8, 2^7)$.

In summary, the overall space complexity is given by

$$S_{\max}^{\text{tot}} = \underbrace{\mathcal{O}(n(n-k))}_{\text{matrices and vectors}} + \underbrace{S_{\max}^{\text{LGA}}}_{\text{the LGA}}. \quad (28)$$

III. PERFORMANCE ANALYSIS OF THE LC-OSD

A. Early Stopping Criteria

The analysis in Sec. II-C suggests that the complexity can be reduced by limiting the maximum number of tests along with properly designed early stopping criteria. Let $\tilde{\mathbf{e}}^{(\ell)} = (\tilde{\mathbf{e}}_L^{(\ell)}, \tilde{\mathbf{e}}_R^{(\ell)})$ be the ℓ -th searched TEP in the LC-OSD algorithm (Algorithm 2) and denote by $\tilde{\mathbf{e}}_{\text{opt}}^{(j)}$ the up-to-date optimal TEP after the j -th iteration. That is,

$$\tilde{\mathbf{e}}_{\text{opt}}^{(j)} = \arg \min_{\tilde{\mathbf{e}}^{(\ell)}: 1 \leq \ell \leq j} \Gamma(\tilde{\mathbf{e}}^{(\ell)}). \quad (29)$$

Example 1. To illustrate the relationship among

$$\tilde{\mathbf{e}}_R^{(\ell)}, \tilde{\mathbf{e}}_L^{(\ell)}, \tilde{\mathbf{e}}^{(\ell)}, \tilde{\mathbf{e}}_{\text{opt}}^{(\ell)}, \quad (30)$$

we have simulated a $\mathcal{C}[128, 64]$ code at $E_b/N_0 = 2.5$ dB. The numerical results are shown in Fig. 1, from which we have the following observation.

- 1) For all ℓ , the heights (soft weights) of the red bar (MRBs) and the green bar (LRBs) add up to that of the small black circle (whole TEP). This is consistent with (23).
- 2) The heights of the red bars keep increasing, which is one of the main features (25) of LC-OSD.
- 3) The height of the blue solid line (soft weight of up-to-date optimal TEP) is non-increasing, and all small black circles are located above the blue line as expected by the definition (29).

Another trivial but not least fact is that whenever the red bar reaches the blue line, i.e.,

$$\Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j)}) \leq \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j)}) \quad (31)$$

for the minimum j ($1 \leq j \leq \ell_{\max}$), the optimal TEP $\tilde{\mathbf{e}}_{\text{opt}}^{(j)}$ is already guaranteed to be ML. This leads to the following trivial stopping criterion.

Definition 2. (*Trivial stopping criterion*) If, for some j , $\Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j)}) \leq \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j)})$, terminate the LC-OSD algorithm and deliver $\mathbf{v} = \mathbf{z} - \tilde{\mathbf{e}}_{\text{opt}}^{(j)} \mathbf{\Pi}^{-1}$ as the output.

Proposition 1. If the LC-OSD is stopped with the trivial stopping criterion being satisfied (usually requiring a sufficiently large ℓ_{\max}), the output is an ML codeword.

Proof: For all $\ell > j$, we have

$$\Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j)}) \leq \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j)}) \quad (\text{by assumption}) \quad (32)$$

$$\leq \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(\ell)}) \quad (\Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j)}) \text{ is non-decreasing}) \quad (33)$$

$$\leq \Gamma(\tilde{\mathbf{e}}^{(\ell)}). \quad (\text{by (23)}) \quad (34)$$

On the other hand, definition (29) ensures that $\Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j)}) \leq \Gamma(\tilde{\mathbf{e}}^{(\ell)})$ also holds for all $\ell \leq j$. Therefore $\tilde{\mathbf{e}}_{\text{opt}}^{(j)}$ is the TEP of an ML codeword. ■

This proposition implies that using the trivial stopping criterion would not incur performance loss, as confirmed numerically in Fig. 1. However, as we can see from the figure, the termination

raised by the trivial stopping criterion is quite behind the location of the true TEP. So, if we could know $\Gamma(\tilde{\mathbf{e}}_L)$ in advance, the LC-OSD can terminate **significantly earlier**. Based on this idea, we propose several early stopping criteria.

Let $\tilde{\mathbf{e}}$ denote the true TEP, i.e.,

$$\tilde{\mathbf{e}} \triangleq \tilde{\mathbf{z}} - \tilde{\mathbf{c}}, \quad (35)$$

where $\tilde{\mathbf{c}} = \mathbf{c}\Pi$ is the permuted **sent** codeword.

Definition 3. (*Ideal stopping criterion*) Assume that $\Gamma(\tilde{\mathbf{e}}_L)$ were known. If for some j , $\Gamma(\tilde{\mathbf{e}}_{opt}^{(j)}) < \Gamma(\tilde{\mathbf{e}}_L) + \Gamma(\tilde{\mathbf{e}}_R^{(j)})$, terminate the LC-OSD algorithm and deliver $\mathbf{v} = \mathbf{z} - \tilde{\mathbf{e}}_{opt}^{(j)}\Pi^{-1}$ as the output.

Proposition 2. If the LC-OSD were stopped with the ideal stopping criterion being satisfied, the performance of the LC-OSD would be no worse (or even better)⁶ than that of the MLD.

Proof: When the LC-OSD is stopped by the ideal stopping criterion, there are two cases.

- The first case is that the true TEP $\tilde{\mathbf{e}}$ has already been searched, i.e., $\tilde{\mathbf{e}}_R = \tilde{\mathbf{e}}_R^{(\ell)}$ for some $\ell \leq j$. In this case, it is not necessary to continue the remaining search.
- The second case is that the true TEP $\tilde{\mathbf{e}}$ has not been searched, indicating that $\Gamma(\tilde{\mathbf{e}}_R) \geq \Gamma(\tilde{\mathbf{e}}_R^{(j)})$. By assumption, it follows that

$$\Gamma(\tilde{\mathbf{e}}_{opt}^{(j)}) < \Gamma(\tilde{\mathbf{e}}_L) + \Gamma(\tilde{\mathbf{e}}_R^{(j)}) \leq \Gamma(\tilde{\mathbf{e}}_L) + \Gamma(\tilde{\mathbf{e}}_R) = \Gamma(\tilde{\mathbf{e}}), \quad (36)$$

which implies that the MLD must be in error and further search is unnecessary either. ■

The ideal criterion is useless for decoding because we do not know $\Gamma(\tilde{\mathbf{e}}_L)$ in advance.

Definition 4. (*Approximate ideal stopping criteria*) If for some j , $\Gamma(\tilde{\mathbf{e}}_{opt}^{(j)}) < \tau + \Gamma(\tilde{\mathbf{e}}_R^{(j)})$, terminate the LC-OSD algorithm, where the threshold τ is a nonnegative number to be determined.

⁶This is not surprising since knowing $\Gamma(\tilde{\mathbf{e}}_L)$ in advance is an ideal but impractical assumption.

We can see from the approximate ideal stopping criteria that the greater the threshold τ , the earlier the termination and hence the lower the complexity. However, a large threshold can lead to premature termination and performance loss. So how large can we set for τ without compromising the performance of LC-OSD? The following proposition provides a bound on the threshold to promise the MLD performance.

Proposition 3. *If the LC-OSD is stopped with the approximate ideal stopping criterion being satisfied and*

$$\tau \leq \Gamma(\tilde{\mathbf{e}}_L) + \Delta\Gamma, \quad (37)$$

then the performance of the LC-OSD is no worse than that of the MLD. Here, the tolerance $\Delta\Gamma$ is positive and defined as

$$\Delta\Gamma \triangleq \begin{cases} \Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_R^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_L^{(j^*)}), & \text{if } j^* > 1 \\ +\infty, & \text{otherwise} \end{cases}, \quad (38)$$

where j^* is the rank of the ML codeword in the search list of LC-OSD.⁷

Proof: Since $\tilde{\mathbf{e}}^{(j^*)}$ is the ML TEP, all preceding TEPs are heavier. Hence, for $j^* > 1$, we have

$$\Delta\Gamma = \Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_R^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_L^{(j^*)}) \quad (39)$$

$$\geq \Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_R^{(j^*)}) - \Gamma(\tilde{\mathbf{e}}_L^{(j^*)}) \quad (40)$$

$$= \Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}^{(j^*)}) \quad (41)$$

$$> 0. \quad (42)$$

The inequality (40) follows from the monotonicity of $\Gamma(\tilde{\mathbf{e}}_R^{(j)})$, the equality (41) follows from (23), and the inequality (42) follows from the hypothesis that j^* is the rank of the ML codeword.

⁷If two or more codewords are ML, we refer to the first one (i.e., $j^* \triangleq \min\{\arg \min_j \Gamma(\tilde{\mathbf{e}}^{(j)})\}$).

When the sent codeword is not ML, the LC-OSD is naturally no worse than the MLD. Therefore, we only need to consider the case when the sent codeword is ML, i.e.,

$$\Gamma(\tilde{\mathbf{e}}) = \Gamma(\tilde{\mathbf{e}}^{(j^*)}). \quad (43)$$

For $j^* = 1$, the ML codeword is always in the list. For $j^* > 1$, we have

$$\Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \tau - \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j^*-1)}) \quad (44)$$

$$\geq \Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \left(\Gamma(\tilde{\mathbf{e}}_{\text{L}}) + \Gamma(\tilde{\mathbf{e}}_{\text{opt}}^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j^*-1)}) - \Gamma(\tilde{\mathbf{e}}_{\text{L}}^{(j^*)}) \right) - \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j^*-1)}) \quad (45)$$

$$= \Gamma(\tilde{\mathbf{e}}_{\text{L}}^{(j^*)}) - \Gamma(\tilde{\mathbf{e}}_{\text{L}}) \quad (46)$$

$$= \left(\Gamma(\tilde{\mathbf{e}}^{(j^*)}) - \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j^*)}) \right) - \left(\Gamma(\tilde{\mathbf{e}}) - \Gamma(\tilde{\mathbf{e}}_{\text{R}}) \right) \quad (47)$$

$$= \Gamma(\tilde{\mathbf{e}}_{\text{R}}) - \Gamma(\tilde{\mathbf{e}}_{\text{R}}^{(j^*)}) \quad (48)$$

$$\geq 0 \quad (49)$$

The equality (48) follows from (43), and the inequality (49) follows from the hypothesis that j^* is the rank of the first ML codeword and the sent is one of the ML codewords. The inequality (49) indicates that the approximate ideal stopping criterion can not be satisfied at $(j^* - 1)$ and before, ensuring the ML codeword is in the search list of the LC-OSD. ■

Proposition 3 suggests that the threshold of the ideal stopping criterion can be increased without degrading the performance. However, the ideal criterion ($\tau = \Gamma(\tilde{\mathbf{e}}_{\text{L}})$) is not helpful in practice, we replace τ by its [expectation \(or approximation\)](#), as presented in the following.

Definition 5. (*Dynamic approximate ideal (DAI) stopping criterion*) *The approximate ideal stopping criterion (Definition 4) with $\tau = \tau_{\text{DAI}}$ is called the DAI stopping criterion, where the DAI threshold is defined by*

$$\tau_{\text{DAI}} \triangleq \mathbb{E}[\Gamma(\tilde{\mathbf{e}}_{\text{L}}) \mid \mathbf{r}]. \quad (50)$$

Now, we show how to calculate the DAI threshold τ_{DAI} . Upon receiving \mathbf{y} , the LLR vector \mathbf{r} can be calculated out by (6). Then, by definition, we have

$$\Gamma(\tilde{\mathbf{e}}_{\text{L}}) = \langle \tilde{\mathbf{e}}_{\text{L}}, |\tilde{\mathbf{r}}_{\text{L}}| \rangle = \sum_{i=1}^{n-k-\delta} \tilde{e}_i |\tilde{r}_i|, \quad (51)$$

and hence

$$\tau_{\text{DAI}} = \mathbb{E}[\Gamma(\tilde{\mathbf{e}}_{\text{L}}) \mid \mathbf{r}] = \mathbb{E}\left[\sum_{i=1}^{n-k-\delta} \tilde{e}_i |\tilde{r}_i| \mid \mathbf{r}\right] = \sum_{i=1}^{n-k-\delta} \mathbb{E}[\tilde{e}_i |\tilde{r}_i| \mid \mathbf{r}] = \sum_{i=1}^{n-k-\delta} \mathbb{E}[\tilde{e}_i \mid \mathbf{r}] \cdot |\tilde{r}_i| \quad (52)$$

The i -th bit of the true TEP \tilde{e}_i is a random variable with

$$\mathbb{P}[\tilde{e}_i = 0 \mid \mathbf{r}] = \frac{1}{1 + \exp(-|\tilde{r}_i|)}, \quad (53)$$

$$\mathbb{P}[\tilde{e}_i = 1 \mid \mathbf{r}] = \frac{1}{1 + \exp(|\tilde{r}_i|)}. \quad (54)$$

Therefore,

$$\tau_{\text{DAI}} = \sum_{i=1}^{n-k-\delta} \frac{1}{1 + \exp(|\tilde{r}_i|)} \cdot |\tilde{r}_i|. \quad (55)$$

Given \mathbf{r} , the threshold τ_{DAI} is computable. The performance of the LC-OSD with the DAI criterion is closely related to whether or not $\tau_{\text{DAI}} \leq \Gamma(\tilde{\mathbf{e}}_{\text{L}}) + \Delta\Gamma$, which in turn depends on the realization of \mathbf{r} . Hence, we have the following proposition from Proposition 3.

Proposition 4. *The LC-OSD with the DAI stopping criterion performs as well as the MLD with a probability of at least $\mathbb{P}[\tau_{\text{DAI}} \leq \Gamma(\tilde{\mathbf{e}}_{\text{L}}) + \Delta\Gamma]$, where the tolerance $\Delta\Gamma$ is defined in Proposition 3.*

Proof: This follows immediately from Proposition 3 by taking into the randomness of the noisy channel (or the LLR vector \mathbf{r}). ■

To reduce the overhead of online calculation, one may use the following stopping criterion instead, which pre-calculates the SNR-dependent thresholds for use.

Definition 6. *(Static approximate ideal (SAI) stopping criterion) The approximate ideal stopping criterion (Definition 4) with $\tau = \tau_{\text{SAI}}$ is called the SAI stopping criterion, where the SAI threshold*

$$\tau_{SAI} \triangleq (n - k - \delta) \mathbb{E}[e|r| \mid |r| \leq \alpha] \quad (56)$$

is the approximation of $\mathbb{E}[\Gamma(\tilde{\mathbf{e}}_L)]$ and α is the $(n - k - \delta)$ -th n -quantile for the random variable $|r|$.

Formally, the $(n - k - \delta)$ -th n -quantile for the random variable $|r|$ can be defined as

$$\alpha \triangleq F_{|r|}^{-1} \left(\frac{n - k - \delta}{n} \right), \quad (57)$$

where $F_{|r|}^{-1}(\cdot)$ is the inverse function of $F_{|r|}(\cdot)$, the cumulative distribution function (CDF) of $|r|$, i.e.,

$$f_{|r|}(r) = \frac{\sigma}{2\sqrt{2\pi}} \exp \left(-\frac{(\sigma^2 r - 2)^2}{8\sigma^2} \right) + \frac{\sigma}{2\sqrt{2\pi}} \exp \left(-\frac{(\sigma^2 r + 2)^2}{8\sigma^2} \right), \quad r \geq 0, \quad (58)$$

$$F_{|r|}(r) = \int_0^r f_{|r|}(t) dt = \frac{1}{2} \operatorname{erf} \left(\frac{\sigma^2 r - 2}{2\sqrt{2}\sigma} \right) + \frac{1}{2} \operatorname{erf} \left(\frac{\sigma^2 r + 2}{2\sqrt{2}\sigma} \right), \quad r \geq 0, \quad (59)$$

and $\operatorname{erf}(\cdot)$ is the error function defined as

$$\operatorname{erf}(x) \triangleq \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt. \quad (60)$$

Therefore, τ_{SAI} (56) can be calculated by

$$\tau_{SAI} = (n - k - \delta) \mathbb{E}[e|r| \mid |r| \leq \alpha] \quad (61)$$

$$= \frac{n - k - \delta}{\mathbb{P}[|r| \leq \alpha]} \int_0^\alpha \frac{r}{1 + \exp(r)} f_{|r|}(r) dr \quad (62)$$

$$= \frac{n - k - \delta}{F_{|r|}(\alpha)} \int_0^\alpha \frac{r}{1 + \exp(r)} f_{|r|}(r) dr \quad (63)$$

$$= n \int_0^\alpha \frac{r}{1 + \exp(r)} f_{|r|}(r) dr \quad (64)$$

From the definition (56), the SAI threshold does not depend on the LLR vector \mathbf{r} . Therefore, to reduce the computational costs, we can calculate offline and store the SAI thresholds for different SNRs. See Table I for an example.

Example 2. For a better understanding of the quantile α , we have provided in Fig. 2 an example

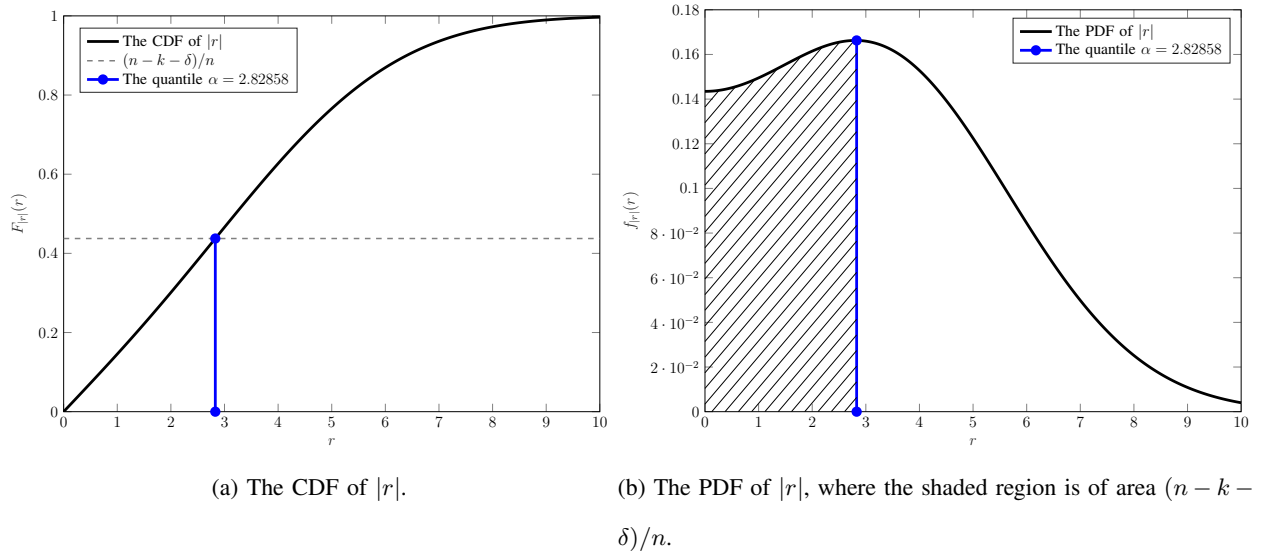


Fig. 2. Sketches of $(n - k - \delta)$ -th n -quantile for $|r|$, where $(n, k, \delta) = (128, 64, 8)$ and $E_b/N_0 = 2.0$ dB.

with the CDF and probability density function (PDF) of $|r|$.

Example 3. To illustrate the accuracy of the approximation, we compare τ_{SAI} with the statistic value of $\mathbb{E}[\Gamma(\tilde{\epsilon}_L)]$ by 10^7 simulations in Table I. We can see from the table that the relative error of τ_{SAI} is small, justifying the negligible performance loss (caused by the SAI stopping).

TABLE I

THE RELATIVE ERROR* OF τ_{SAI} WITH DIFFERENT COMBINATION OF (n, k, δ) .

(n, k, δ)	(128, 32, 8)		(128, 32, 12)		(128, 64, 8)		(128, 64, 12)		(128, 92, 8)		(128, 92, 12)	
E_b/N_0	τ_{SAI}	Error	τ_{SAI}	Error	τ_{SAI}	Error	τ_{SAI}	Error	τ_{SAI}	Error	τ_{SAI}	Error
0.0 dB	18.34	+0.18%	17.30	+0.16%	12.07	+0.44%	11.04	+0.40%	4.528	-0.03%	3.431	-0.68%
1.0 dB	18.91	+0.27%	17.98	+0.27%	12.34	+0.60%	11.45	+0.62%	5.081	+0.80%	3.983	+0.35%
2.0 dB	19.01	+0.33%	18.23	+0.34%	11.96	+0.66%	11.29	+0.72%	5.348	+1.51%	4.396	+1.45%
3.0 dB	18.48	+0.33%	17.88	+0.37%	10.78	+0.56%	10.36	+0.66%	5.031	+1.70%	4.382	+2.08%
4.0 dB	17.19	+0.30%	16.79	+0.34%	8.880	+0.36%	8.668	+0.43%	4.033	+1.26%	3.718	+1.84%

*Here, the relative error is defined to $(\tau_{\text{SAI}} - \mathbb{E}[\Gamma(\tilde{\epsilon}_L)]) / (\mathbb{E}[\Gamma(\tilde{\epsilon}_L)])$

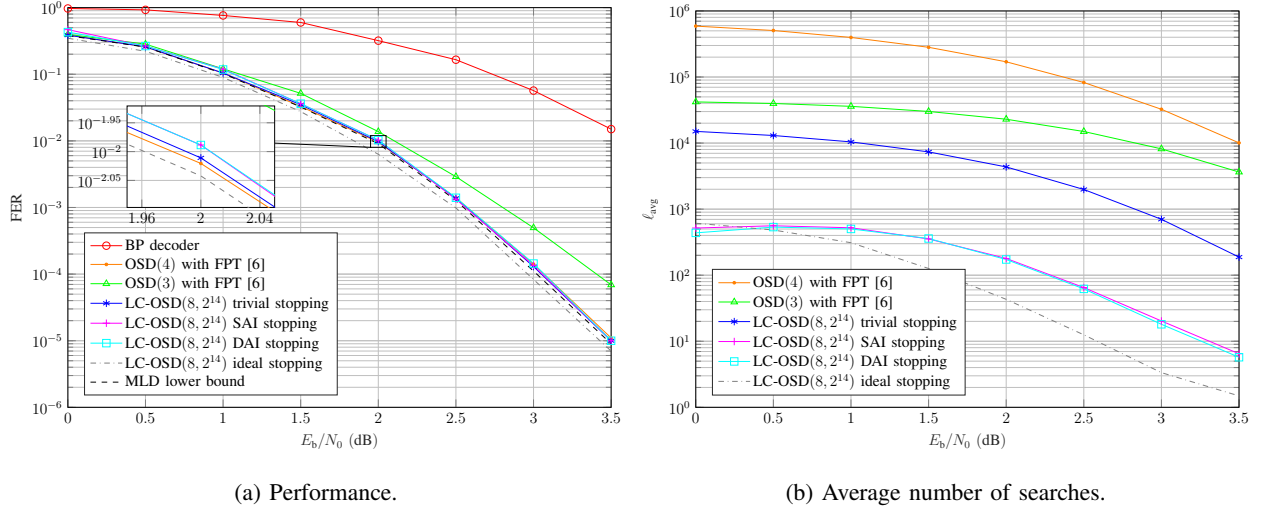


Fig. 3. Simulation results of the LDPC code $\mathcal{C}_{\text{LDPC}}[128, 64]$ constructed in [19].

Example 4. Consider the rate-1/2 LDPC code $\mathcal{C}_{\text{LDPC}}[128, 64]$ constructed in [19]. The constraint degree $\delta = 8$ and the maximum number of searches $\ell_{\max} = 2^{14}$. Several decoders are compared in terms of the frame error rate (FER) and the average search number, as shown in Fig. 3, where the OSD with FPT [6] using trivial stopping criterion is a low-complexity implementation for OSD that has the same performance as OSD. The MLD lower bound in Fig. 3a is a simulated bound as presented in [20] and calculated by utilizing the LC-OSD (see Algorithm 3).

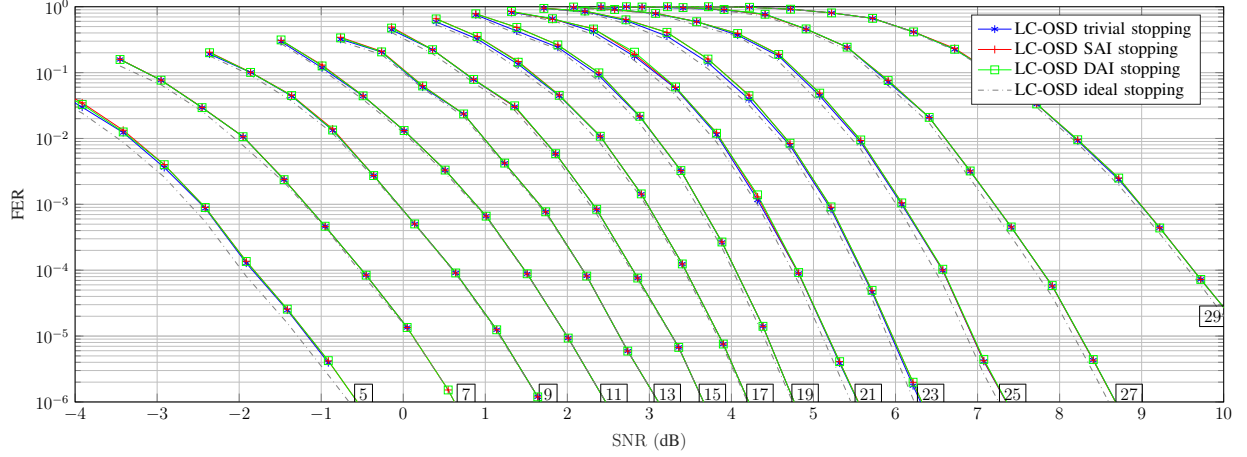
Algorithm 3 Lower bound on error counter for MLD

Input: \mathbf{H} , δ , ℓ_{\max} , transmission-receiving pair (\mathbf{c}, \mathbf{y})

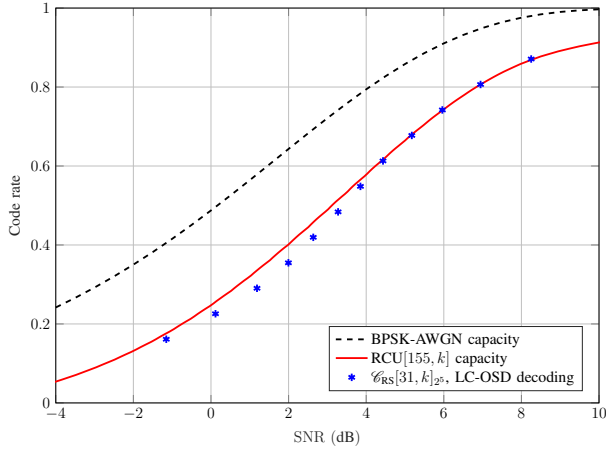
Output: MLD error is detected or not.

-
- 1: **function** MLD-ERROR-COUNTER($\mathbf{H}, \delta, \ell_{\max}, \mathbf{c}, \mathbf{y}$)
 - 2: $\hat{\mathbf{c}} \leftarrow \text{LC-OSD}(\mathbf{H}, \mathbf{y}, \delta, \ell_{\max})$ \triangleright Search for the optimal codeword by the LC-OSD.
 - 3: **return** $f_{\mathbf{y}|\mathbf{c}}(\mathbf{y} | \hat{\mathbf{c}}) > f_{\mathbf{y}|\mathbf{c}}(\mathbf{y} | \mathbf{c})$ \triangleright An MLD error is detected if $\hat{\mathbf{c}}$ is more likely than \mathbf{c} .
-

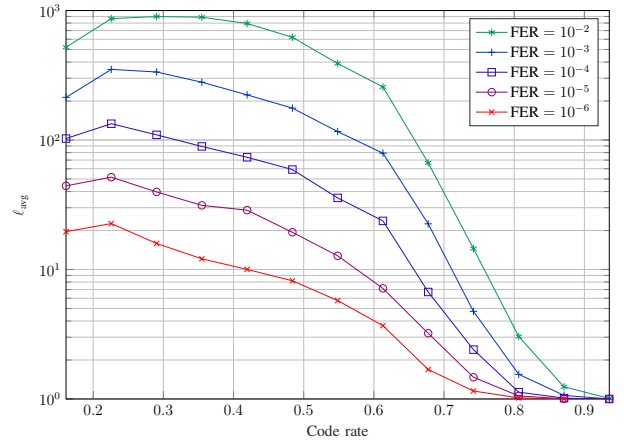
Example 5. To demonstrate the effect of various criteria on the LC-OSD, we present the



(a) Performance. The box k with the dimension k inside is marked for each result cluster.



(b) The finite-length capacity and the required SNR by the RS codes with the LC-OSD. Here, the LC-OSD is with the DAI criterion, and the target FER is 10^{-5} .



(c) Average number of searches of LC-OSD with the DAI criterion at the different FERs (corresponding to different SNRs for different code rates).

Fig. 4. Simulation results of the RS codes $\mathcal{C}_{\text{RS}}[31, k]_{2^5}$, which are mapped into \mathbb{F}_2^{155} for LC-OSD(8, 2^{14}).

simulation results for Reed-Solomon (RS) codes with different code rates in Fig. 4. The simulated RS codes are defined over the field \mathbb{F}_{2^5} , denoted by $\mathcal{C}_{\text{RS}}[31, k]_{2^5}$ with k as the dimension. They are mapped into \mathbb{F}_2^{155} and decoded by the LC-OSD. The performances are shown in Fig. 4a, from which we see that the approximate ideal stopping (including SAI and DAI) criteria perform as well as the trivial stopping criterion. In Fig. 4b, we compare the RS codes with the RCU

bounds, from which we can see that the achieving code rates of the RS codes are close to the RCU bounds. This suggests that the RS codes are good codes in the finite length region. Further, we present the average list size ℓ_{avg} for different code rates and target FER. As shown in Fig. 4b, when the RS codes run on their working regions ($\text{FER} \approx 10^{-5}$), the average numbers of searches are reduced to tens or even a few. Equipped with the tailored stopping criteria, LC-OSD is indeed a sub-ML decoding algorithm for short block codes. In Fig. 4c, we show the average number of searches of LC-OSD at different FERs, which correspond to different SNRs for different code rates. For example, to achieve $\text{FER} = 10^{-4}$, the required SNRs range from about -1.75 dB for the lowest code rate to about 9.5 dB for the highest code rate. The results show that, in the low FER region, the average number of searches can be very low, suggesting potential applications in future communications where the target reliability is 99.99999%.

For the reason that the LC-OSD with the DAI stopping criterion has much lower time complexity without incurring noticeable performance loss⁸, we use the LC-OSD with the DAI stopping criterion for simulations in the rest of this paper.

B. Influences of Decoding Parameters

Recalling that $\delta = 0$ implies the original OSD algorithm and $\delta = n - k$ implies the MLD, we can safely infer that, as the increase of δ , the maximum number of searches can be reduced. Therefore, in this subsection, we discuss the relationship between FER and the decoding parameters $(\delta, \ell_{\text{max}})$.

1) *The Channel of MRBs*: The channel of each MRB is nearly⁹ an “order statistic” AWGN channel

$$\tilde{y}_i = \tilde{x}_i + \tilde{w}_i, \quad n - k - \delta < i \leq n. \quad (65)$$

⁸This was also verified by simulations in [7] and holds for a wide class of short codes.

⁹This is because the linear dependency of columns of \mathbf{H} and hence $|\tilde{\mathbf{r}}|$ is not always non-decreasing.

Formally, $|\tilde{y}_i|$ ($n - k - \delta < i \leq n$) is the i -th order statistic of $|y_i|$ [2], [17], whose probability density function given by

$$f_{|\tilde{y}_i|}(y) = \frac{n!}{(i-1)!(n-i)!} f_{|y|}(y) (F_{|y|}(y))^{i-1} (1 - F_{|y|}(y))^{n-i}, \quad (66)$$

where $f_{|y|}(\cdot)$ and $F_{|y|}(\cdot)$ are the probability density function and cumulative distribution function of $|y|$, respectively, i.e.,

$$f_{|y|}(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) + \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right), \quad y \geq 0, \quad (67)$$

$$F_{|y|}(y) = \int_0^y f_{|y|}(t) dt = \frac{1}{2} \operatorname{erf}\left(\frac{y-1}{\sqrt{2}\sigma}\right) + \frac{1}{2} \operatorname{erf}\left(\frac{y+1}{\sqrt{2}\sigma}\right), \quad y \geq 0. \quad (68)$$

2) *The MRB Code:* After preprocessing, the MRB part of the code $\mathcal{C}[n, k]$ is determined.

Definition 7. (MRB code) The MRB code $\mathcal{C}_{\text{MRB}}[k + \delta, k]$ is a code from puncturing $\mathcal{C}[n, k]$, where the LRBs (see Sec. II) are punctured.

Proposition 5. Up to a column permutation, the local-constraint matrix \mathbf{P}_2 (see Sec. II) is a parity-check matrix of the MRB code.

Proof: The proof follows immediately from (15). ■

From the Definition 7 and Proposition 5, we can see that the MRB code is a semi-random code due to the randomness of the LLR vector \mathbf{r} .

In the following, instead of $\tilde{\mathbf{e}}_{\text{R}}$, $\tilde{\mathbf{y}}_{\text{R}}$, $\tilde{\mathbf{r}}_{\text{R}}$ and $\tilde{\mathbf{z}}_{\text{R}}$, we use for simplicity the notations \mathbf{e} , \mathbf{y} , \mathbf{r} and \mathbf{z} to denote the MRB part of the TEP, the received vector, the LLR vector, and the bit-wise hard decision vector, respectively.

Denote the number of MRB codewords lighter than the sent MRB codeword by L , i.e.,

$$L \triangleq \left| \left\{ \mathbf{v} \in \mathcal{C}_{\text{MRB}} : \Gamma(\mathbf{f}) < \Gamma(\mathbf{e}), \mathbf{f} = \mathbf{z} - \mathbf{v} \right\} \right|. \quad (69)$$

Then the FER of the MRB code under the optimal list decoder¹⁰ with the list size of ℓ_{\max} is

¹⁰Here, the optimal list decoder generates a list containing the ℓ_{\max} lightest TEPs.

given by

$$\varepsilon_{\text{MRB}} \triangleq \mathbb{P}[L \geq \ell_{\max}]. \quad (70)$$

3) The Random Code and Performance Approximation:

Definition 8. (Random code) Let $\hat{\mathcal{C}}$ be a multi-subset of \mathbb{F}_2^N with 2^K elements. If all elements of $\hat{\mathcal{C}}$ are independently and uniformly drawn from \mathbb{F}_2^N , then we call $\hat{\mathcal{C}}$ as a random code, denoted as $\hat{\mathcal{C}}[N, K]$.

Similarly, the list FER performance of the random code, denoted $\hat{\mathcal{C}}_{\text{MRB}}[k + \delta, k]$, is defined as

$$\hat{\varepsilon}_{\text{MRB}} \triangleq \mathbb{P}[\hat{L} \geq \ell_{\max}], \quad (71)$$

where \hat{L} denotes the number of codeword lighter the sent codeword in $\hat{\mathcal{C}}_{\text{MRB}}$, i.e.,

$$\hat{L} \triangleq |\{\mathbf{v} \in \hat{\mathcal{C}}_{\text{MRB}} : \Gamma(\mathbf{f}) < \Gamma(\mathbf{e}), \mathbf{f} = \mathbf{z} - \mathbf{v}\}|. \quad (72)$$

Here, we introduce a set

$$\mathcal{D} \triangleq \{\mathbf{f} \in \mathbb{F}_2^{k+\delta} : \Gamma(\mathbf{f}) < \Gamma(\mathbf{e})\} \quad (73)$$

of all sequences lighter than the correct TEP \mathbf{e} and denote its cardinality as

$$D \triangleq |\mathcal{D}| = \sum_{\mathbf{f} \in \mathbb{F}_2^{k+\delta}} \mathbb{I}[\Gamma(\mathbf{f}) < \Gamma(\mathbf{e})], \quad (74)$$

where $\mathbb{I}[\cdot]$ is the indicator function. Due to the symmetry, we can safely assume that the all-zero codeword $\mathbf{0}$ (of \mathcal{C}_{MRB} or $\hat{\mathcal{C}}_{\text{MRB}}$) is sent.

The cardinality D of the set \mathcal{D} depends only on the LLR vector \mathbf{r} , denoted $D(\mathbf{r})$, which does not rely on any specific code. One way to calculate $D(\mathbf{r})$ is to directly count the number by listing all vectors in the set \mathcal{D} with algorithms such as the FPT (Sec. IV-B). This is suitable for short codes ($n \leq 64$) but is time-consuming for longer codes. In the latter case, we use the saddlepoint method to approximate the cardinality $D(\mathbf{r})$.

To do so, we first introduce a probability model over $\mathbb{F}_2^{k+\delta}$. Formally, let \mathbf{f} be a random sequence distributed uniformly over $\mathbb{F}_2^{k+\delta}$. That is, the probability of any $\mathbf{f}^* \in \mathbb{F}_2^{k+\delta}$ is

$$\mathbb{P}[\mathbf{f} = \mathbf{f}^*] = 2^{-(k+\delta)}. \quad (75)$$

With this probability model, we can rewrite the cardinality as

$$D(\mathbf{r}) = 2^{k+\delta} \cdot \mathbb{P}[\Gamma(\mathbf{f}) < \Gamma(\mathbf{e}) | \mathbf{r}] \quad (76)$$

$$= 2^{k+\delta} \cdot \mathbb{P} \left[\sum_{i=1}^{k+\delta} f_i |r_i| < \sum_{i=1}^{k+\delta} e_i |r_i| \middle| \mathbf{r} \right] \quad (77)$$

$$= 2^{k+\delta} \cdot \mathbb{P} \left[\sum_{i=1}^{k+\delta} (f_i - e_i) |r_i| < 0 \middle| \mathbf{r} \right] \quad (78)$$

$$= 2^{k+\delta} \cdot \mathbb{P} \left[\sum_{i=1}^{k+\delta} W_i < 0 \middle| \mathbf{r} \right] \quad (79)$$

$$= 2^{k+\delta} \cdot \mathbb{P}[W < 0 | \mathbf{r}], \quad (80)$$

where $W_i = (f_i - e_i)|r_i|$ and $W = \sum_{i=1}^{k+\delta} W_i$. Recalling that $e_i = \mathbb{I}[r_i < 0]$, we can show that W_i ($1 \leq i \leq k + \delta$) are independent random variables with probabilities

$$\mathbb{P}[W_i = 0 | \mathbf{r}] = \mathbb{P}[f_i = e_i | \mathbf{r}] = 1/2, \quad (81)$$

$$\mathbb{P}[W_i = r_i | \mathbf{r}] = \mathbb{P}[f_i \neq e_i | \mathbf{r}] = 1/2. \quad (82)$$

Therefore, the calculation of the cardinality is equivalent to evaluating the tail probability, which can be solved by saddlepoint approximation [12], [13]. To be precise, let $\kappa(s)$ be the cumulant generating function of the summation W , then

$$\kappa(s) \triangleq \log \mathbb{E}[e^{sW}] \quad (83)$$

$$= \sum_{i=1}^{k+\delta} \log \mathbb{E}[e^{sW_i}] \quad (84)$$

$$= \sum_{i=1}^{k+\delta} \log \left(\frac{1}{2} + \frac{1}{2} e^{sr_i} \right). \quad (85)$$

By using saddlepoint method [13], (given \mathbf{r} .) the PDF of W can be approximated by

$$f_W(w) \approx \exp(\kappa(\hat{s}) - \hat{s}w) \cdot \frac{1}{\sqrt{2\pi\kappa''(\hat{s})}} \exp\left(-\frac{(w - \kappa'(\hat{s}))^2}{2\kappa''(\hat{s})}\right), \quad (86)$$

where κ' (or κ'') is the first (or second) derivative of κ and \hat{s} is the solution¹¹ of $\kappa'(s) = 0$.

Therefore, the cardinality can be approximated by

$$D(\mathbf{r}) = 2^{k+\delta} \cdot \mathbb{P}[W < 0 | \mathbf{r}] \quad (87)$$

$$= 2^{k+\delta} \cdot \int_{-\infty}^0 f_W(w) dw \quad (88)$$

$$\approx 2^{k+\delta} \cdot \frac{1}{2} \exp\left(\kappa(\hat{s}) - \hat{s}\kappa'(\hat{s}) + \frac{1}{2}\hat{s}^2\kappa''(\hat{s})\right) \cdot \operatorname{erfc}\left(\frac{\kappa'(\hat{s}) - \hat{s}\kappa''(\hat{s})}{\sqrt{2\kappa''(\hat{s})}}\right), \quad (89)$$

where $\kappa(s)$ is defined in (83) and $\operatorname{erfc}(\cdot)$ is the complementary error function defined as

$$\operatorname{erfc}(x) \triangleq 1 - \operatorname{erf}(x). \quad (90)$$

Example 6. In this example, we employ two techniques to get the complementary CDF (CCDF) of $D(\mathbf{r})$, formulated by

$$F_D^c(d) \triangleq \mathbb{P}[D(\mathbf{r}) > d], \quad (91)$$

in which \mathbf{r} is a random vector generated by simulations. The first method utilizes the FPT method (Sec. IV-B) to count the exact number of sequences in \mathcal{D} . The second method employs the saddlepoint approach presented in (89). The simulation results are shown in Fig. 5, from which we see that the results from the saddlepoint method match well with those from the counting. We also see that, as E_b/N_0 increases, $D(\mathbf{r})$ decreases in statistical sense, as expected. For example, the rank of the true TEP is less than 10^2 for about 70% realizations of \mathbf{r} at $E_b/N_0 = 2$ dB while about 90% at $E_b/N_0 = 3$ dB.

¹¹It can be proven that $\kappa'(s)$ is an increasing function and has exactly one root if $(\max_i r_i)(\min_i r_i) < 0$.

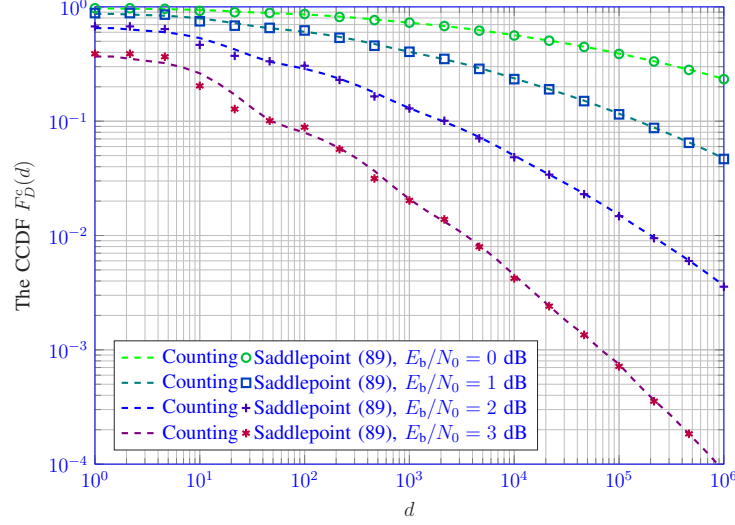


Fig. 5. The CCDF of $D(\mathbf{r})$, where the randomness comes from the LLR vector \mathbf{r} . Here $(n, k, \delta) = (128, 64, 8)$.

Once the cardinality of \mathcal{D} is obtained, \hat{L} is randomly distributed as the binomial distribution.

$$\hat{L} \sim \text{Binomial}(D(\mathbf{r}), 2^{-\delta}) \quad (92)$$

with the probability mass function (PMF) given by

$$\mathbb{P}[\hat{L} = \ell \mid D(\mathbf{r})] = \binom{D(\mathbf{r})}{\ell} (2^{-\delta})^\ell (1 - 2^{-\delta})^{D(\mathbf{r}) - \ell} \quad (93)$$

Hence, we can derive the FER of the random code $\hat{\mathcal{C}}_{\text{MRB}}$ as follows.

$$\hat{\varepsilon}_{\text{MRB}} = \mathbb{P}[\hat{L} \geq \ell_{\max}] = \mathbb{E}_{\mathbf{r}}[\mathbb{P}[\hat{L} \geq \ell_{\max} \mid D(\mathbf{r})]]. \quad (94)$$

Example 7. In this example, we present in Fig. 6 the list decoding FERs¹² of four types of codes: CA-polar, RM, eBCH, and RBL codes. The results show that the list decoding FERs for different codes are well estimated by the proposed approximation (94).

¹²In this paper, the list decoding FER is defined as the probability of the sent codeword not presented in the list, denoted ε_{MRB} .

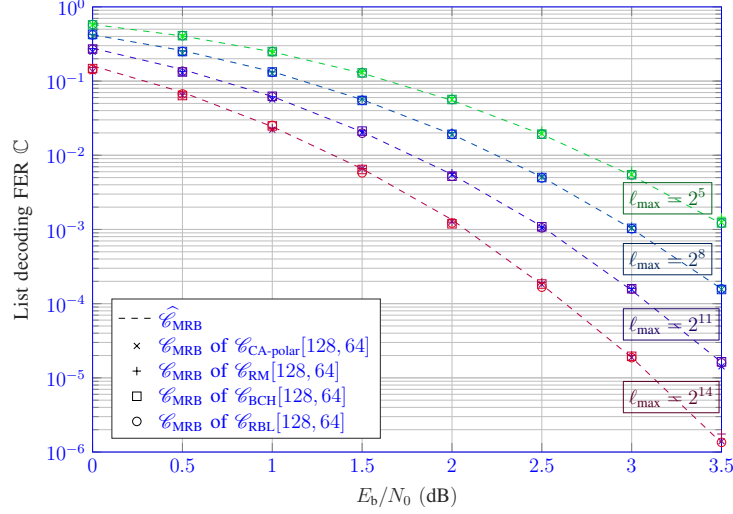


Fig. 6. List decoding FERs ε_{MRB} of CA-polar, RM, eBCH, and RBL codes (markers) with varying list size ℓ_{max} , comparing with the estimation by random code (dashed lines), where $(n, k, \delta) = (128, 64, 8)$.

Remarks. It is worth pointing out that the above analysis, starting from the definition (71) of $\widehat{\varepsilon}_{\text{MRB}}$, does not depend on any specific code. Since we do not have a closed-form expression for the distribution of $D(\mathbf{r})$, the approximation (94) still relies on simulating \mathbf{r} . However, the proposed analysis approach is very different from the conventional simulations, at least in the sense that first, the simulation does not require decoding, and second, the simulation results do not rely on any specific codes but are applicable to analyzing all specific codes with the same code length and code rate, as confirmed by Example 7.

Also notice that, ε_{MRB} and $\widehat{\varepsilon}_{\text{MRB}}$ are functions of ℓ_{max} , denoted as $\varepsilon_{\text{MRB}}(\ell_{\text{max}})$ and $\widehat{\varepsilon}_{\text{MRB}}(\ell_{\text{max}})$ respectively. Then we can write the PMF of L (and \widehat{L}) by

$$\mathbb{P}[L = \ell] = \mathbb{P}[L \geq \ell] - \mathbb{P}[L \geq \ell + 1] = \varepsilon_{\text{MRB}}(\ell) - \varepsilon_{\text{MRB}}(\ell + 1), \quad (95)$$

$$\mathbb{P}[\widehat{L} = \ell] = \mathbb{P}[\widehat{L} \geq \ell] - \mathbb{P}[\widehat{L} \geq \ell + 1] = \widehat{\varepsilon}_{\text{MRB}}(\ell) - \widehat{\varepsilon}_{\text{MRB}}(\ell + 1). \quad (96)$$

Example 8. To illustrate the accuracy of the approximation, we use $\mathbb{P}[L = \ell] \approx \mathbb{P}[\widehat{L} = \ell]$ to estimate the PMF of L and compare it with the statistic value by simulating two different

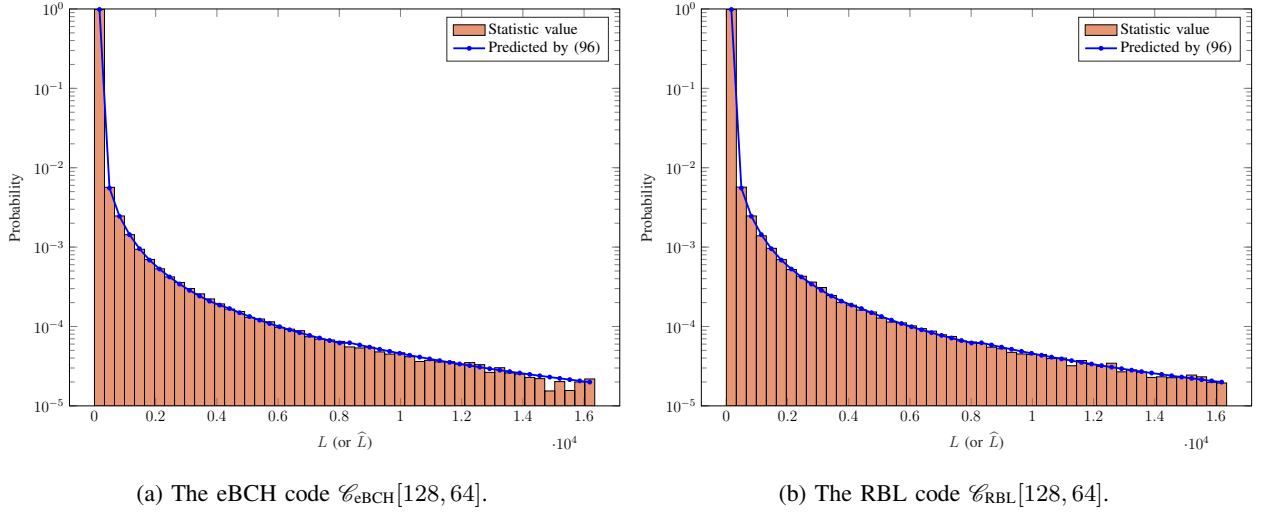


Fig. 7. The statistical histogram of L , the number of codewords better than the sent codeword, in the case of $\delta = 8$ and $E_b/N_0 = 2.0$ dB.

codes, an extended Bose-Chaudhuri-Hocquenghem (eBCH) code $\mathcal{C}_{\text{eBCH}}[128, 64]$ (Fig. 7a) and a random binary linear (RBL) code $\mathcal{C}_{\text{RBL}}[128, 64]$ whose parity-check matrix is randomly constructed (Fig. 7b). As seen from Fig. 7, the statistic values are quite close to the predicted values from (71) and (96) and are less relevant to the structure of the code. Hence, we have $\mathbb{E}[L] \approx \mathbb{E}[\hat{L}]$ and use $\hat{\varepsilon}_{\text{MRB}}$ to estimate ε_{MRB} in the following analysis.

Conditional on the transmitted codeword being in the list, we have

$$\mathbb{E}[L \mid L < \ell_{\max}] \approx \mathbb{E}[\hat{L} \mid \hat{L} < \ell_{\max}], \quad (97)$$

where the RHS can be calculated as

$$\mathbb{E}[\hat{L} \mid \hat{L} < \ell_{\max}] = \frac{1}{1 - \hat{\varepsilon}_{\text{MRB}}(\ell_{\max})} \left(\sum_{\ell=1}^{\ell_{\max}-1} \hat{\varepsilon}_{\text{MRB}}(\ell) - (\ell_{\max} - 1) \cdot \hat{\varepsilon}_{\text{MRB}}(\ell_{\max}) \right). \quad (98)$$

Example 9. We have simulated the left-hand side (LHS) of (97) of the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$ as well as the RBL code $\mathcal{C}_{\text{RBL}}[128, 64]$. The results are shown in Fig. 8. Also shown in Fig. 8 are the numerical values calculated by (98). We can see that the simulated values are close to

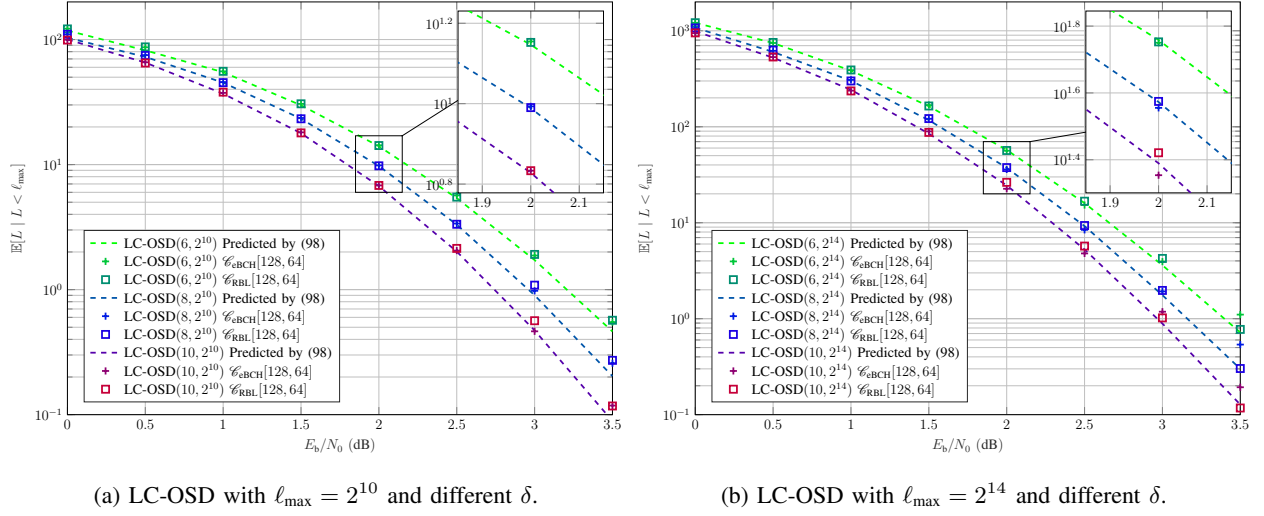


Fig. 8. Conditional expectation of L of the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$ and the RBL code $\mathcal{C}_{\text{RBL}}[128, 64]$ along with their predictions.

each other and also to the calculated values, validating the assumption on the randomness of \mathcal{C}_{MRB} .

4) *Performance Bound:* Denote by ε and ε_{ML} the FER of the LC-OSD and MLD, respectively.

Then we have

$$\varepsilon = \mathbb{P}[\{e \text{ is not in the list}\}] + \mathbb{P}[\{e \text{ is in the list but is not the most likely one}\}] \quad (99)$$

$$\leq \mathbb{P}[\{e \text{ is not in the list}\}] + \mathbb{P}[\{e \text{ is not the most likely one}\}] \quad (100)$$

$$= \varepsilon_{\text{MRB}} + \varepsilon_{\text{ML}} \quad (101)$$

Hence, the performance gap between the LC-OSD and the MLD can be approximately upper bounded by $\hat{\varepsilon}_{\text{MRB}}$ as

$$\varepsilon - \varepsilon_{\text{ML}} \leq \varepsilon_{\text{MRB}} \approx \hat{\varepsilon}_{\text{MRB}} \quad (102)$$

which does not depend on the structure of the code.

Example 10. Fig. 9 shows the performance of the RBL code $\mathcal{C}_{\text{eBCH}}[128, 64]$. We also plotted

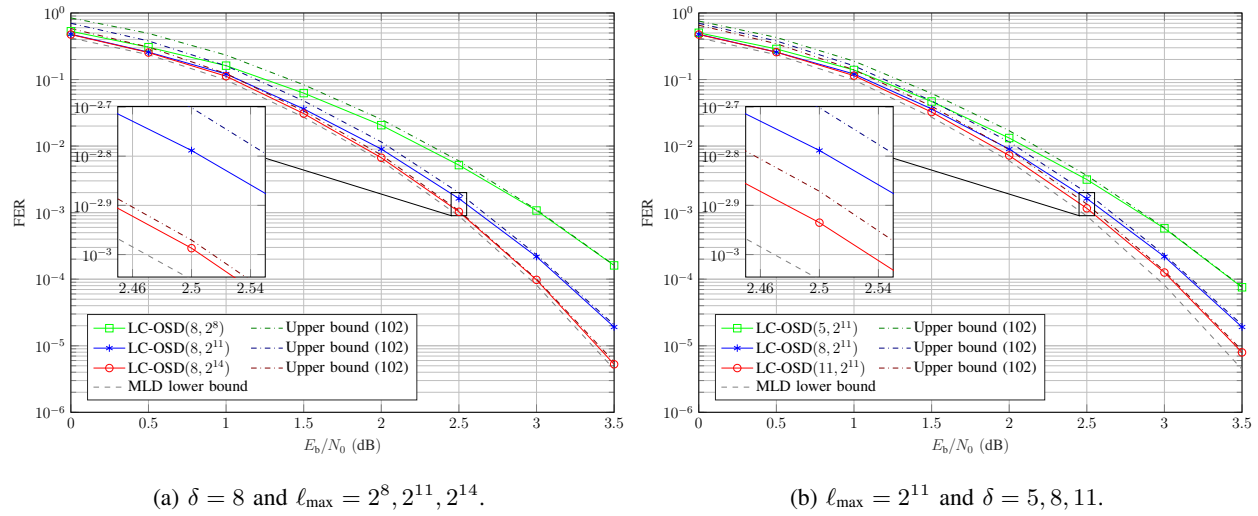


Fig. 9. Performance and approximate upper bound of the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$.

the approximate upper bound (102) of $\mathcal{C}_{\text{eBCH}}[128, 64]$. The values of $\hat{\epsilon}_{\text{MRB}}$ in this example are calculated by the saddlepoint approximation discussed in Example 6. As shown in Fig. 9 we can observe that the performance of $\mathcal{C}_{\text{eBCH}}[128, 64]$ improves as ℓ_{\max} or δ increases. Additionally, we note that the approximate upper bounds are highly accurate due to the accuracy of (71), which has already been verified in Example 8 and Example 9.

C. Decoding Parameters Tuning

The tight performance upper bound deduced in Sec. III-B4 for the LC-OSD becomes a helpful tool in many situations. In this subsection, we will give examples that explain how we could tune the decoding parameters (δ, ℓ_{\max}) to satisfy the diversified requirements depending on application scenarios.

To evaluate the time consumption of the LC-OSD, we need to write out the time factor (in

the case when the SLVA (see Sec. IV-A for details) is taken as the LGA) (26) and (27), namely,

$$T_{\text{avg}}^{\text{tot}} \approx \rho_1 \cdot n(n-k)(n-k-\delta) + \rho_2 \cdot 2^\delta(k+\delta) + \rho_3 \cdot \ell_{\text{avg}}(n-k-\delta)(k+\delta), \quad (103)$$

$$T_{\text{max}}^{\text{tot}} \approx \rho_1 \cdot n(n-k)(n-k-\delta) + \rho_2 \cdot 2^\delta(k+\delta) + \rho_3 \cdot \ell_{\text{max}}(n-k-\delta)(k+\delta), \quad (104)$$

where ρ_i ($i = 1, 2, 3$) are called the time factor constants. We can measure these time factor constants by implementing the LC-OSD on a specific computer (or device).

Example 11. Table II shows the values of these time factor constants for reference¹³.

TABLE II
REFERENCE VALUES OF TIME FACTORS.

Time factor constants	ρ_1	ρ_2	ρ_3
Measured values (ns)	0.0816	26.4	0.728

Then, we may tune the decoding parameters as required by applications.

1) Minimum Average Time Complexity: In the scenarios where the power consumption is concerned, we need to optimize $(\delta, \ell_{\text{max}})$ to minimize the average time complexity $T_{\text{avg}}^{\text{tot}}$.

Example 12. We have simulated the average number of searches of the LC-OSD for the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$ using different decoding parameters $(\delta, \ell_{\text{max}})$. From Fig. 10a, we see that the worse the channel is, the higher complexity of the LC-OSD is, as in line with our intuition. We have implemented the LC-OSD algorithm with SLVA on the desktop computer and tried to optimize the speed as much as possible.

2) Minimum Worst-case Time Complexity: In the scenarios where the maximum delay is concerned, we need to limit the maximum list size ℓ_{max} to control the worst-case decoding time.

¹³We have measured these values on a desktop computer with a CPU of 11th Gen Intel(R) Core(TM) i7-11700F @ 2.50GHz.

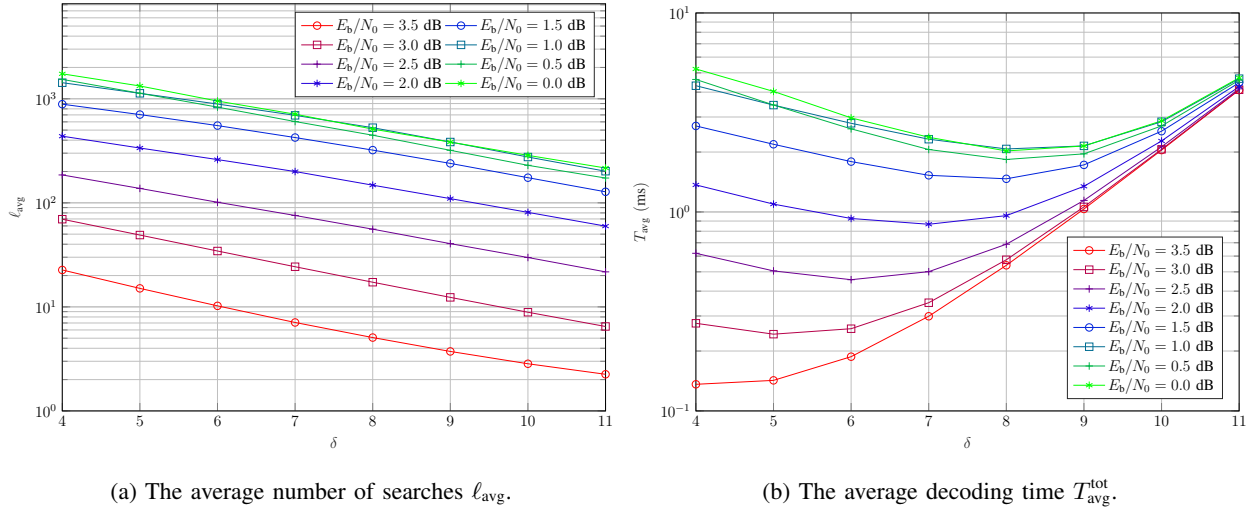


Fig. 10. Average time complexity of the LC-OSD with $\ell_{\text{max}} = 2^{14}$ for the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$.

Recalling the definition in (71), $\widehat{\varepsilon}_{\text{MRB}}$ is a function of $(\delta, \ell_{\text{max}})$, denoted as $\widehat{\varepsilon}_{\text{MRB}}(\delta, \ell_{\text{max}})$. Intuitively, $\widehat{\varepsilon}_{\text{MRB}}(\delta, \ell_{\text{max}})$ is a non-increasing function of ℓ_{max} for a fixed δ . Therefore, given a target FER $\widehat{\varepsilon}_{\text{MRB}}^*$, let $\ell_{\text{max}}^*(\delta)$ be the minimum ℓ_{max} for a fixed δ under the constraint of the target FER, i.e.,

$$\ell_{\text{max}}^*(\delta) \triangleq \min_{\widehat{\varepsilon}_{\text{MRB}}(\delta, \ell_{\text{max}}) \leq \widehat{\varepsilon}_{\text{MRB}}^*} \ell_{\text{max}}. \quad (105)$$

To meet the requirement of minimum worst-case time complexity, we can adjust the decoding parameter to $(\ell_{\text{max}}^*, \delta)$.

Example 13. Intuitively, as δ increases, the constraint specified by \mathbf{P}_2 becomes more “global”, and hence, the number of searches required for the LC-OSD to have near MLD performance decreases. In the extreme case when δ reaches its maximum value $(n - k)$, $\ell_{\text{max}} = 1$ is sufficient. This intuition can be verified by the simulation. Fig. 11a shows the required ℓ_{max} versus δ for the LC-OSD to have near MLD performance. As expected, the larger δ is, the smaller ℓ_{max} is required.

Example 14. At first glance, it seems that a larger δ has higher decoding efficiency. However, this

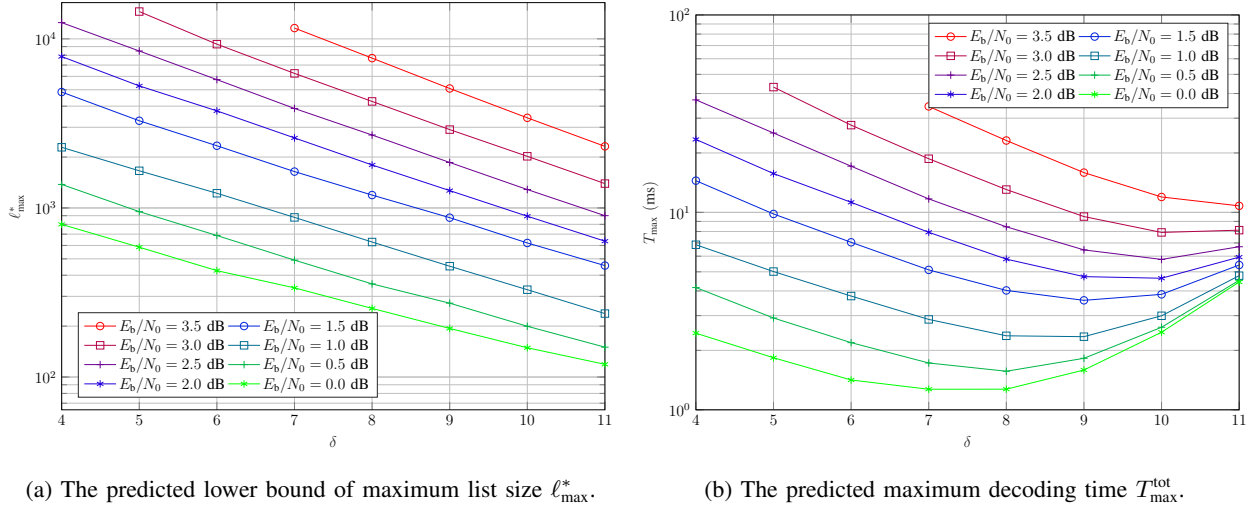


Fig. 11. Tuning parameters on (105) for the LC-OSD of the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$ to have near MLD performance ($\hat{\varepsilon}_{\text{MRB}}^* = \text{FER}_{\text{ML}}$).

is not always the case. On the one hand, smaller δ values lead to larger ℓ_{\max} values, resulting in slower searches. On the other hand, large δ values (which result in small ℓ_{\max}) can have significant delays caused by initialization (as discussed in Sec. II-C1). As a result, the maximum time complexity is dominated by search when δ is small and by initialization when δ is large. Therefore, it is meaningful to carefully tune the values of δ and ℓ_{\max} for achieving high decoding efficiency. This has been verified by the simulation shown in Fig. 11b.

3) *Limited Maximum Space Complexity:* In the scenarios where the maximum memory usage is concerned, we need to consider the problem of parameter tuning in the case of limited space. The examples in these cases are similar to the previous scenarios and will not be repeated here.

IV. TWO LGAS FOR LC-OSD

The above analysis for LC-OSD is independent of the LGA. In practice, we need an algorithm that can generate MRB sequences one by one in the order of increasing soft weight (see Sec. II-B). In this section, we provide two LGAs for LC-OSD, which are applicable to general

linear block codes and hence to the MRB code (see Definition 7) with length N and dimension K .

A. List Viterbi Decoding Algorithm

The list Viterbi decoding algorithms (LVAs) (including the parallel LVA, see, for example, [21], and the serial LVA) produce an ordered list of the ℓ best candidates.

1) *Trellis and Viterbi Algorithm:* A code (block or convolutional) can be represented graphically by a trellis [22]–[24], which makes it possible (also convenient) to implement the MLD with reduced complexity. The most well-known trellis-based MLD algorithm is the Viterbi algorithm.

A binary linear code $\mathcal{C}[N, K]$ with a parity-check matrix \mathbf{P} can be represented by a trellis with N sections and at most 2^{N-K} states at each section. The state space at level- i , written as $\Sigma_i(\mathcal{C})$, is the set of allowable states at the i -th level. In the trellis, every vertex at level- i has a path from the starting state. Every vertex in the trellis has at least one incoming edge except for the starting state and at most two outgoing edges except for the states at the last level. For $i > 0$, the two outgoing edges from a state at the $(i - 1)$ -th level represent the bit 0 and bit 1, leading the state $\sigma \in \Sigma_{i-1}(\mathcal{C})$ to $(\sigma + \mathbf{0}) \in \Sigma_i(\mathcal{C})$ and $(\sigma + \mathbf{P}_i) \in \Sigma_i(\mathcal{C})$, respectively, where \mathbf{P}_i is the i -th column of \mathbf{P} .

Once a trellis of a code is constructed, the ML codeword could be obtained by running the Viterbi algorithm over the trellis. Equivalently, the Viterbi algorithm can be implemented to find the shortest path (lightest error pattern) \mathbf{e} from the starting state $\mathbf{0}$ to the ending state $\mathbf{P}\mathbf{z}^T$ over the trellis, where the soft weights of the dashed edges are 0 and that of the solid edges are the bit reliability $|r_i|$. See the discussion in Sec. II.

2) *Parallel LVA:* The parallel LVA (PLVA) searches for the ℓ_{\max} best paths at the same time by updating the ℓ_{\max} best paths into each state level by level. **Therefore, the time complexity of the PLVA is**

$$T_{\text{PLVA}} = \mathcal{O}(\ell_{\max} 2^{N-K} N), \quad (106)$$

which could be significantly reduced by the “calculate-when-requested” technique presented in below.

3) *Serial LVA*: The serial LVA (SLVA) searches for the ℓ_{\max} most probable candidates, one by one, starting with the most probable path. The main advantage of this algorithm is that the ℓ -th best path is only computed when the previously found $\ell - 1$ paths are not good enough, such as not fulfilling the [global](#) constraints. More importantly, after the first path is calculated, the SLVA can produce each following path in linear time and space complexity, which is done by using the intermediate results from previous calculations. This avoids many unnecessary computations of the PLVA.

Here, we present a recursive implementation of SLVA (Algorithm 4)¹⁴. The algorithm first allocates an array to store paths and costs for all trellis vertexes. Then, the algorithm visits the ending state vertex and queries the ℓ -th best path. When visiting a vertex $\sigma \in \Sigma_i(\mathcal{C})$, the algorithm is going to visit recursively the two incoming vertexes (predecessors $\sigma \in \Sigma_{i-1}(\mathcal{C})$ and $\sigma - \mathbf{P}_i \in \Sigma_{i-1}(\mathcal{C})$) and select the better one. If an unallowable state vertex is reached, it returns a path with the cost of $+\infty$. Finally, the ℓ -th best path at the ending state vertex is presented after a recursive search.

When $\ell \geq 2$, the vertex only requests the predecessor on the $(\ell - 1)$ -th path to generate a new candidate for comparison with the unused but available candidate at the other predecessor. For example, let vertexes A and B be the predecessors of C . The first path to C (denoted $[\overset{1\text{st}}{\rightsquigarrow} C]$) is obtained by comparing $[\overset{1\text{st}}{\rightsquigarrow} A \rightarrow C]$ and $[\overset{1\text{st}}{\rightsquigarrow} B \rightarrow C]$. Say $[\overset{1\text{st}}{\rightsquigarrow} A \rightarrow C]$ is better, then $[\overset{1\text{st}}{\rightsquigarrow} C] = [\overset{1\text{st}}{\rightsquigarrow} A \rightarrow C]$. If required, $[\overset{2\text{nd}}{\rightsquigarrow} C]$ is obtained by comparing $[\overset{2\text{nd}}{\rightsquigarrow} A \rightarrow C]$ (newly generated candidate) and $[\overset{1\text{st}}{\rightsquigarrow} B \rightarrow C]$ (unused but available candidate). This is illustrated in

¹⁴The basic idea of the SLVA was initially presented by Seshadri and Sundberg in [14]. The description in Sec. IV-A is based on Wenchao Lin’s thesis [25], which is believed to be more general than that given in [14].

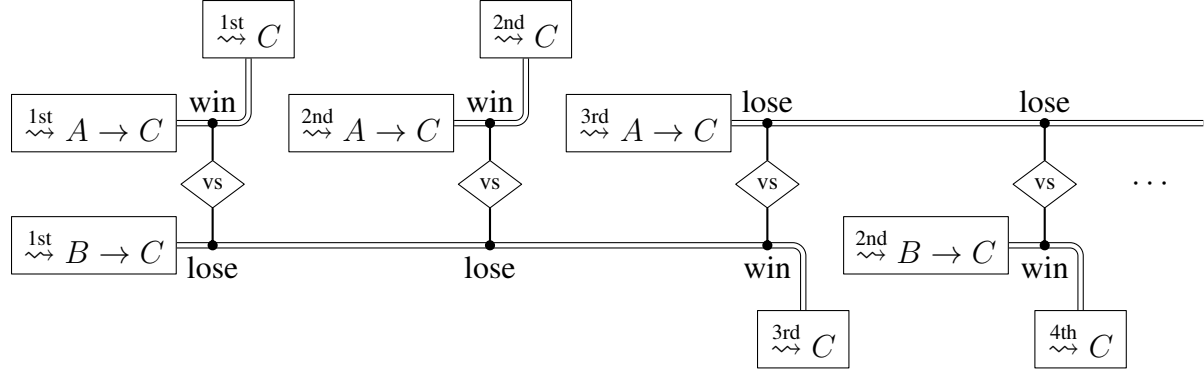


Fig. 12. A process of selecting ℓ best paths for vertex C , which is assumed to have two predecessors A and B . The notation $[j\text{-th path to } X]$ stands for the j -th path to X and then one step to C .

Fig. 12. Therefore, the time complexity of the SLVA is

$$T_{\text{avg}}^{\text{SLVA}} = \underbrace{\mathcal{O}(2^{N-K}N)}_{\text{the first candidate}} + \underbrace{\mathcal{O}((\ell_{\text{avg}} - 1)N)}_{\text{other candidates}} = \mathcal{O}((2^{N-K} + \ell_{\text{avg}})N). \quad (107)$$

Algorithm 4 Serial list Viterbi Algorithm (SLVA)

Input: The parity-check matrix $\mathbf{P} \in \mathbb{F}_2^{(N-K) \times N}$, the reliability vector $|\mathbf{r}| \in \mathbb{R}^N$, the ending state $\mathbf{s}_{\text{end}} \in \mathbb{F}_2^{N-K}$, the rank of TEP to search ℓ .

Output: The ℓ -th lightest TEP $\mathbf{e}^{(\ell)}$ that satisfies the parity-check constraints.

```

1:  $nodes \leftarrow$  a sequence array of  $N \times |\mathbb{F}_2^{N-K}|$  of  $(path, cost)$  pairs  $\triangleright$  Allocate globally.
2: function SLVA( $\mathbf{P}, |\mathbf{r}|, \mathbf{s}_{\text{end}}, \ell$ )
3:    $\mathbf{s}_{\text{start}} \leftarrow \mathbf{0}$   $\triangleright$  Set the starting state.
4:   function SLVA-IMPL( $i, \mathbf{s}, \ell$ )  $\triangleright$  A recursive implementation.
5:     if  $i = 0$  then  $\triangleright$  Boundary of recursion.
6:       if  $\mathbf{s} = \mathbf{s}_{\text{start}}$  and  $\ell = 1$  then  $\triangleright$  The starting state  $\mathbf{s}_{\text{start}}$  has only 1 legal path.
7:         return  $(\varepsilon, 0)$   $\triangleright$  Empty string and initial cost.
8:       return  $(\varepsilon, +\infty)$   $\triangleright$  Infinity cost for unreachable states.
9:     while  $\text{length}(nodes[i][\mathbf{s}]) < \ell$  do  $\triangleright$  Check whether it is calculated or not.
10:      for  $b = 0, 1$  do  $\triangleright$  enumerate the incoming edges.
11:         $\mathbf{s}_b \leftarrow \mathbf{s} - b \cdot \mathbf{P}_i^T$   $\triangleright$  Minus operator is defined in vector space  $\mathbb{F}_2^{N-K}$ .
12:         $t_b \leftarrow$  the number of paths end with  $b$  in  $nodes[i][\mathbf{s}]$ 
13:         $(path_b, cost_b) \leftarrow$  SLVA-IMPL( $i - 1, \mathbf{s}_b, t_b + 1$ )  $\triangleright$  Recursive search.
14:         $cost_b \leftarrow cost_b + b \cdot |r_i|$   $\triangleright$  Cost accumulation.
15:         $path_b \leftarrow path_b \parallel b$   $\triangleright$  Append a single bit.
16:         $b^* \leftarrow \arg \min_{b \in \{0,1\}} cost_b$   $\triangleright$  The optimal transition.
17:         $nodes[i][\mathbf{s}] \leftarrow nodes[i][\mathbf{s}] \parallel (path_{b^*}, cost_{b^*})$   $\triangleright$  Append to information sequence.
18:      return  $nodes[i][\mathbf{s}][\ell]$   $\triangleright$  The  $\ell$ -th information of  $nodes[i][\mathbf{s}]$ .
19:     $(\mathbf{e}^{(\ell)}, cost) \leftarrow$  SLVA-IMPL( $N, \mathbf{s}_{\text{end}}, \ell$ )
20:  return  $\mathbf{e}^{(\ell)}$ 

```

Example 15. To better understand the SLVA, we present as an example to show how to generate

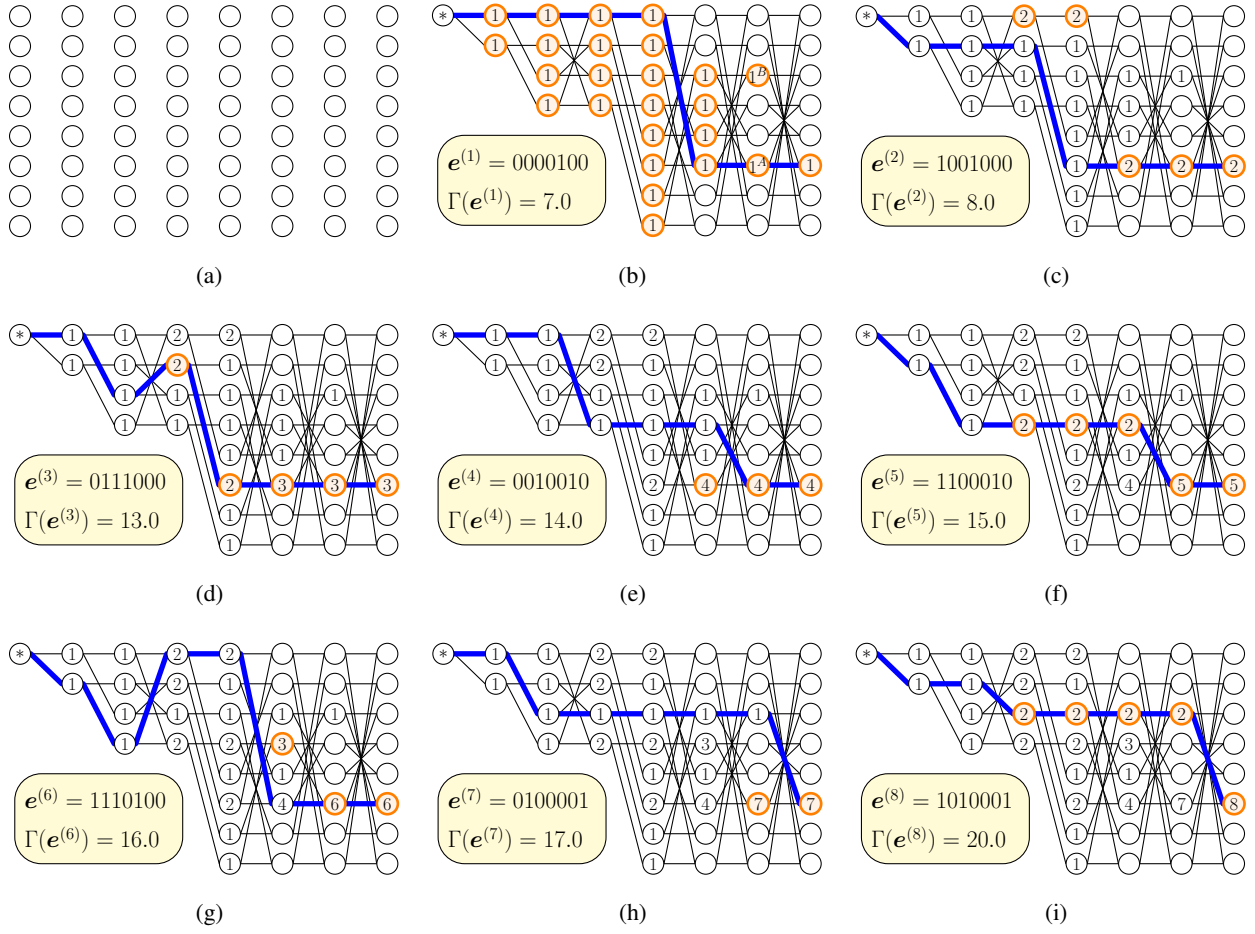


Fig. 13. An illustration example of SLVA over a trellis for the $[7, 4]$ Hamming code. The digit in each circle counts the number of candidates, while empty circles imply no candidate. The updated states are highlighted in orange.

candidates through the trellis defined by the parity check matrix of the $(7, 4)$ Hamming code

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (108)$$

The trellis representation¹⁵ of the Hamming code is shown in Fig. 13b. It is worth noting that the trellis does not correspond to the Hamming code but its cosets.

¹⁵The edges are directed from left to right, while the arrows are omitted for the sake of clarity.

Suppose that the LLR vector is

$$\mathbf{r} = [-2.0, 3.0, 4.0, -6.0, 7.0, 10.0, 14.0], \quad (109)$$

resulting in the bit-wise hard-decision vector

$$\mathbf{z} = [1, 0, 0, 1, 0, 0, 0]. \quad (110)$$

Initially, the space of size $2^{N-K} \times N$ for the trellis is “blanked”, meaning that no branches exist between the states, as shown in Fig. 13a. Then the syndrome, i.e., the ending state of the trellis, is given by

$$\mathbf{s}_{\text{end}} \triangleq \mathbf{z}\mathbf{P}^T = [1, 0, 1]. \quad (111)$$

By the SLVA, the best eight candidates $e^{(i)}, 1 \leq i \leq 8$, are generated serially, shown in the yellow boxes along with their soft weights, and highlighted in blue paths through the trellis in Figs. 13b-13i. The first candidate $e^{(1)}$ is generated by the VA with the complexity of order $\mathcal{O}(2^{N-K}N)$, which starts from the initial state $\mathbf{s}_{\text{start}} = [0, 0, 0]$ and implicitly generates all (prefix) best paths, labeled by $\textcircled{1}$ in the orange circles. Now, to generate the second (best) path, the ending state compares the two paths, the second path entering the state $\textcircled{1^A}$ and the first path entering the state $\textcircled{1^B}$ (since $\textcircled{1^B}$ has not been used). That is, the state $\textcircled{1^A}$ needs to generate the second path, while the state $\textcircled{1^B}$ does not have to. Then, the state $\textcircled{1^A}$ takes on the same role as the ending state. This process continues recursively and requires the complexity of order $\mathcal{O}(N)$. To avoid updating unnecessary states, the SLVA uses the same processes to generate the subsequent candidates, reducing the time complexity compared to the PLVA.

We conclude (without proof) that when searching for the ℓ -th candidate ($\ell > 1$), the states to be updated must lay on the $(\ell - 1)$ -th path. Therefore, searching for the ℓ -th candidate consumes only $\mathcal{O}(N)$ time whenever $\ell > 1$.

Example 16. Fig. 14 shows that the performance of the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$ improves as

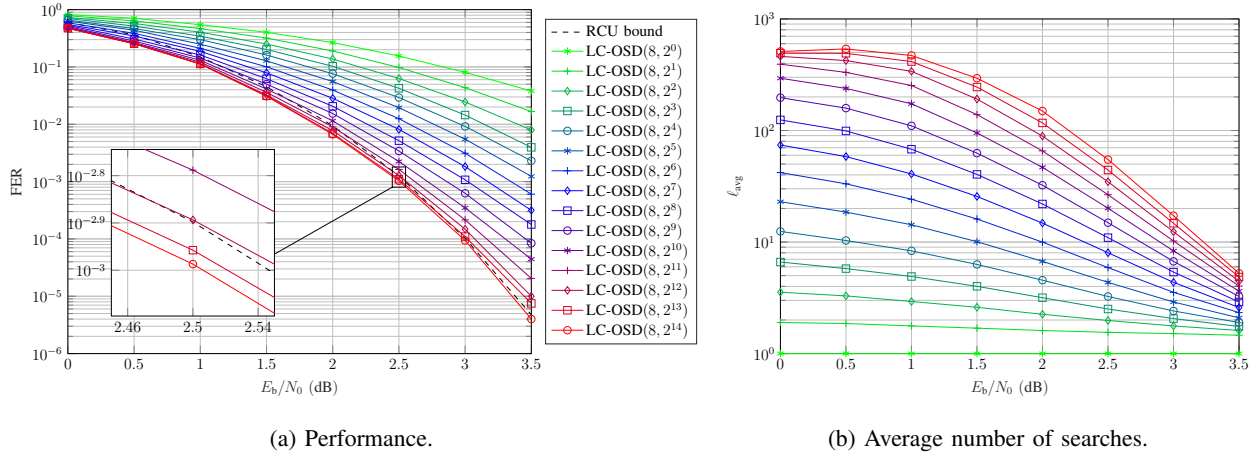


Fig. 14. Simulation results of the eBCH code $\mathcal{C}_{\text{eBCH}}[128, 64]$.

ℓ_{\max} increases but saturates at a certain level of ℓ_{\max} . This is reasonable because the performance of the LC-OSD is lower bound by that of the MLD, regardless of how large ℓ_{\max} is.

B. Flipping Pattern Tree and Its Variants

This subsection presents the flipping pattern tree (FPT) algorithm and its variants.

1) *Introduction of FPT*: FPT is a tree taking vectors in \mathbb{F}_2^N as vertexes; the FPT algorithm is an algorithm that runs on the FPT to generate an ordered list of the ℓ best candidates. Without causing ambiguity, we also refer to the FPT algorithm simply as the FPT. Unlike the LVA, the candidates of the FPT are not constrained. Formally, let N be the candidate's length, and $|\mathbf{r}| \in \mathbb{R}^N$ is the reliability vector. Without loss of generality, we assume that the reliability vector is non-decreasing, i.e.,

$$|r_1| \leq |r_2| \leq \dots \leq |r_N|. \quad (112)$$

The FPT can output one candidate at a time (and at most 2^N candidates in total), with non-decreasing soft weight, i.e.,

$$\Gamma(\mathbf{e}^{(1)}) \leq \Gamma(\mathbf{e}^{(2)}) \leq \dots \leq \Gamma(\mathbf{e}^{(\ell)}) \leq \dots \leq \Gamma(\mathbf{e}^{(2^N)}). \quad (113)$$

A brute-force implementation of FPT is to generate all 2^N candidates, sort them by soft weights in ascending order, and output them one by one. This brute-force method is suitable for small N but does not work for large N due to its exponential complexity. To circumvent such a brute-force search, we give the following definition.

Definition 9. (*Precedence*)¹⁶ For two TEP $e, e' \in \mathbb{F}_2^N$, we say e precedes e' (denoted $e \preceq e'$) if and only if there exists an injection $\pi : \text{supp}(e) \rightarrow \text{supp}(e')$ such that $i \leq \pi(i)$, where the support $\text{supp}(\cdot)$ is defined as

$$\text{supp}(e) \triangleq \{i \in \{1, \dots, N\} : e_i \neq 0\}. \quad (114)$$

Here, the injection π is called a witness of $e \preceq e'$.

Proposition 6. If for all positive integer j , the inequality

$$|\text{supp}_{\geq j}(e)| \leq |\text{supp}_{\geq j}(e')|. \quad (115)$$

holds, then $e \preceq e'$ with a witness π that maps the t -th greatest element of $\text{supp}(e)$ to the t -th greatest element of $\text{supp}(e')$. To be precise, let

$$\text{supp}(e) = \{i_1, i_2, \dots, i_w\}, \quad w = |\text{supp}(e)|, \quad i_t > i_{t+1} \quad (1 \leq t < w), \quad (116)$$

$$\text{supp}(e') = \{i'_1, i'_2, \dots, i'_{w'}\}, \quad w' = |\text{supp}(e')|, \quad i'_t > i'_{t+1} \quad (1 \leq t < w'). \quad (117)$$

Then $\pi(i_t) = i'_t$ ($1 \leq t \leq w$).

Here $\text{supp}_{\geq j}(\cdot)$ is defined as

$$\text{supp}_{\geq j}(e) \triangleq \{i \in \text{supp}(e) : i \geq j\}. \quad (118)$$

Proof: By substituting $j = 1$ in (115), we know that $w \leq w'$ and hence π is a well-defined injection. For $1 \leq t \leq w$, we have $t = |\text{supp}_{\geq i_t}(e)| \leq |\text{supp}_{\geq i_t}(e')|$, implying that $i'_t \in \text{supp}_{\geq i_t}(e')$ and hence $i_t \leq i'_t = \pi(i_t)$. Therefore, π is a witness of $e \preceq e'$. ■

¹⁶This is investigated in constructions of polar codes, such as [26] and [27].

Taking into account the fact that

$$\pi(\text{supp}_{\geq j}(e)) \subseteq \text{supp}_{\geq j}(e'), \quad (119)$$

we see that Proposition 6 is actually an equivalent definition for “ \preceq ” (Definition 9). That is,

$$e \preceq e' \iff \forall j \in \mathbb{Z}^+ (|\text{supp}_{\geq j}(e)| \leq |\text{supp}_{\geq j}(e')|). \quad (120)$$

Proposition 7. $(\mathbb{F}_2^N, \preceq)$ is a partial order set.

Proof: Let $e^{(a)}, e^{(b)}, e^{(c)} \in \mathbb{F}_2^N$.

- 1) Reflexivity. The identity map is a witness of $e^{(a)} \preceq e^{(a)}$.
- 2) Transitivity. Assume $\pi^{(ab)}$ and $\pi^{(bc)}$ are witnesses of $e^{(a)} \preceq e^{(b)}$ and $e^{(b)} \preceq e^{(c)}$, respectively. Let $\pi^{(ac)} \triangleq \pi^{(bc)}\pi^{(ab)}$. Then $\pi^{(ac)} : \text{supp}(e^{(a)}) \rightarrow \text{supp}(e^{(c)})$ is an injection. By Definition 9, we have

$$i \leq \pi^{(ab)}(i) \leq \pi^{(bc)}(\pi^{(ab)}(i)) = \pi^{(ac)}(i). \quad (121)$$

Therefore $\pi^{(ac)}$ is a witness of $e^{(a)} \preceq e^{(c)}$.

- 3) Antisymmetry. Assume $\pi^{(ab)}$ and $\pi^{(ba)}$ are witnesses of $e^{(a)} \preceq e^{(b)}$ and $e^{(b)} \preceq e^{(a)}$, respectively. Let $\pi^{(aa)} \triangleq \pi^{(ba)}\pi^{(ab)}$. Then $\pi^{(aa)} : \text{supp}(e^{(a)}) \rightarrow \text{supp}(e^{(a)})$ is a bijection. Moreover, because $\pi^{(aa)}$ is a witness of $e^{(a)} \preceq e^{(a)}$, $\pi^{(aa)}$ must be an identity map (which can be proven by induction). By Definition 9, we have

$$i \leq \pi^{(ab)}(i) \leq \pi^{(ba)}(\pi^{(ab)}(i)) = \pi^{(aa)}(i) = i. \quad (122)$$

Hence, $\pi^{(ab)}(i) = i$ and thus $\text{supp}(e^{(a)}) \subseteq \text{supp}(e^{(b)})$. Analogously, $\text{supp}(e^{(b)}) \subseteq \text{supp}(e^{(a)})$.

Therefore, $\text{supp}(e^{(a)}) = \text{supp}(e^{(b)})$ and thus $e^{(a)} = e^{(b)}$.

■

Example 17. Let $N = 4$. We have

$$0000 \preceq 1000 \preceq 0100 \preceq 0010 \preceq 0001, \quad (123)$$

$$1000 \preceq 1100 \preceq 1010 \preceq 1001 \preceq 0111, \quad (124)$$

$$1001 \not\preceq 0110, 0110 \not\preceq 1001, \quad (125)$$

$$0001 \not\preceq 1010, 1010 \not\preceq 0001, \quad (126)$$

$$0000 \preceq \mathbf{e} \preceq 1111, \forall \mathbf{e} \in \mathbb{F}_2^N. \quad (127)$$

Proposition 8. If $\mathbf{e} \preceq \mathbf{e}'$ and $|\mathbf{r}|$ is a non-decreasing vector, then $\Gamma(\mathbf{e}) \leq \Gamma(\mathbf{e}')$.

Proof: Let π be a witness of $\mathbf{e} \preceq \mathbf{e}'$. By Definition 1, we have

$$\Gamma(\mathbf{e}) = \sum_{i \in \text{supp}(\mathbf{e})} |r_i| \leq \sum_{i \in \text{supp}(\mathbf{e})} |r_{\pi(i)}| = \sum_{j \in \pi(\text{supp}(\mathbf{e}))} |r_j| \leq \sum_{j \in \text{supp}(\mathbf{e}')} |r_j| = \Gamma(\mathbf{e}'). \quad (128)$$

■

Proposition 8 says that the precedence preserves the order of soft weights. Conversely, we have the following proposition.

Proposition 9. If $\mathbf{e} \not\preceq \mathbf{e}'$, then there exists a non-decreasing vector $|\mathbf{r}|$ such that $\Gamma(\mathbf{e}) > \Gamma(\mathbf{e}')$.

Proof: From Proposition 6, we know that there exists a positive integer j such that

$$|\text{supp}_{\geq j}(\mathbf{e})| > |\text{supp}_{\geq j}(\mathbf{e}')|. \quad (129)$$

Assigning

$$r_i = \begin{cases} 0, & \text{if } i < j \\ 1, & \text{if } i \geq j \end{cases}, \quad (130)$$

we have $\Gamma(\mathbf{e}) = |\text{supp}_{\geq j}(\mathbf{e})| > |\text{supp}_{\geq j}(\mathbf{e}')| = \Gamma(\mathbf{e}')$. ■

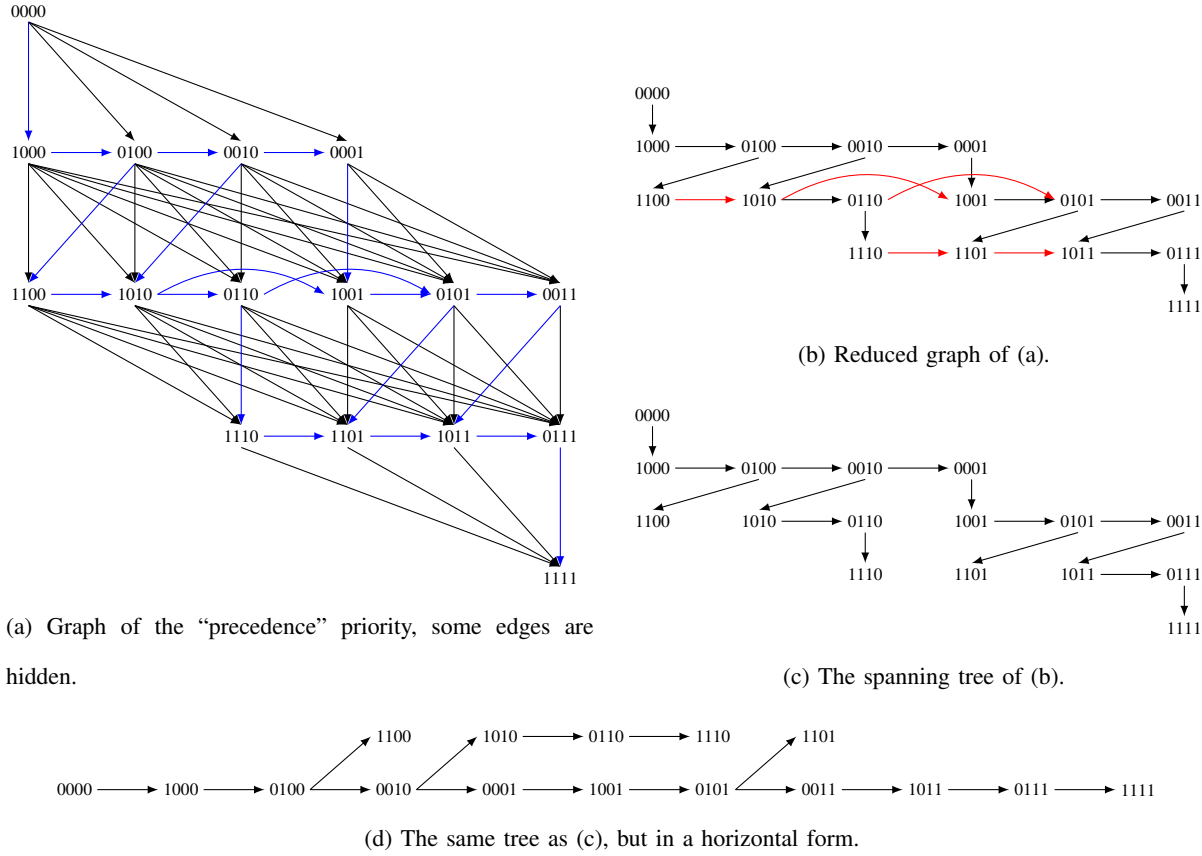


Fig. 15. Graph representations of the FPT.

If one can build a graph (see Fig. 15a for example) on the “precedence” partial order, i.e., take TEPs as vertexes and “ \preceq ” as directed edges, one can generate candidates through the topological sort algorithm [28]. The time complexity is generally proportional to the average degree of the graph, which is $(2^N - 1)/2$ in our case (the proof is omitted here). For reducing the complexity (or the average degree), one can equivalently run the topological sort on the reduced graph (see Fig. 15b for example).

However, the reduced graph has an average degree of $(N + 1)/4$ (the proof is omitted here). If we build a spanning tree (see Fig. 15c) from the reduced graph, the average degree can be reduced to less than one. Here, we give the method of systematically building a spanning tree

from the reduced graph on the precedence partial order.

Definition 10. (*Flipping pattern tree*)¹⁷ The flipping pattern tree (FPT) is a directed graph with TEPs as vertexes and edges defined in the following. For a TEP e , denote $\mathcal{S} \triangleq \text{supp}(e)$. If $1 \notin \mathcal{S}$, the TEP e has an out-edge to the TEP with $\mathcal{S}_L(e)$ as the support set, where

$$\mathcal{S}_L(e) \triangleq \mathcal{S} \cup \{1\}; \quad (131)$$

if $\mathcal{S} \neq \emptyset$, $\min \mathcal{S} < N$ and $(\min \mathcal{S} + 1) \notin \mathcal{S}$, the TEP e has an out-edge to the TEP with $\mathcal{S}_R(e)$ as the support set, where

$$\mathcal{S}_R(e) \triangleq \mathcal{S} \setminus \{\min \mathcal{S}\} \cup \{\min \mathcal{S} + 1\}. \quad (132)$$

We will see in the next example (Example 18) that the flipping pattern tree is a rooted binary tree with the all-zero TEP as the root. For a TEP e with $\mathcal{S} = \text{supp}(e)$, we may refer to the TEP specified by \mathcal{S}_L as the left child and \mathcal{S}_R as the right child.

Example 18. From Definition 10, we see that the left child (if it exists) of a TEP is generated by setting its *leftmost position* to 1, and the right child (if it exists) is generated by shifting its *leftmost one* to the right. Therefore, we can classify the TEPs into several types by inspecting their parent and children. From the table below (Table III), we can conclude that the FPT takes $(00 \cdots 0)$ as root, vectors of the form $(11X \cdots X)$ as leaves, and the remaining vectors as inner vertexes. An inner vertex has two children if and only if it is in the form of $(00 \cdots 010X \cdots X)$, i.e., its leftmost one occurs in the context of the form 010.

¹⁷This idea of building a tree for TEPs was proposed in [20] (with an earlier version [29]) for decoding of RS codes and was applied to semi-LDPC-CCs [30]. Similar trees for binary codes were also constructed in [5, Algorithm 1] and [31, Algorithm 2]. Because the FPT is a spanning tree (Fig. 15c) of the reduced graph (Fig. 15b) instead of the full graph (Fig. 15a), a careful inspection (the details are omitted here) can find that the tree in this paper is the most effective in memory load.

TABLE III
PARENT AND CHILDREN FOR DIFFERENT TYPES* OF TEPs.

Type pattern	(00...0)	(00...01X...X)			(1X...X)	
Parent	—	(00... <u>1</u> 0X...X)			(<u>0</u> X...X)	
Type pattern	(00...0)	(00...01)	(00...010X...X)	(00...011X...X)	(10X...X)	(11X...X)
Left child	(<u>1</u> 0...0)	(<u>1</u> 0...01)	(<u>1</u> 0...010X...X)	(<u>1</u> 0...011X...X)	—	—
Right child	—	—	(00...0 <u>0</u> 1X...X)	—	(<u>0</u> 1X...X)	—

*Here, ‘X’ is a placeholder for either 0 or 1.

Based on the FPT, we proposed the FPT algorithm (Algorithm 5). In fact, the FPT algorithm is the specialization of topological sort on the FPT.

Algorithm 5 Flipping pattern tree (FPT) algorithm

Input: the ascending sorted reliability vector $|r| \in \mathbb{R}^N$, the rank of TEP to search ℓ .

Output: The ℓ -th lightest TEP $e^{(\ell)}$.

```

1:  $\mathcal{Q} \leftarrow \{0\}$   $\triangleright$  A priority queue allocated globally.
2: function FPT( $|r|, \ell$ )  $\triangleright$  This should be called in order of  $\ell = 1, 2, \dots, \ell_{\max}$ .
3:    $e^{(\ell)} \leftarrow \arg \min_{e \in \mathcal{Q}} \Gamma(e)$ 
4:    $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{e^{(\ell)}\}$ 
5:   if  $\mathcal{S}_L(e^{(\ell)})$  exists then  $\triangleright$  Left child, defined in (131).
6:   |    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\text{the TEP specified by } \mathcal{S}_L(e^{(\ell)})\}$ 
7:   if  $\mathcal{S}_R(e^{(\ell)})$  exists then  $\triangleright$  Right child, defined in (132).
8:   |    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\text{the TEP specified by } \mathcal{S}_R(e^{(\ell)})\}$ 
9:   return  $e^{(\ell)}$ 

```

The time complexity of line 3 is proportional to $\log |\mathcal{Q}|$ if we use the priority queue as the data structure. Since the FPT is a binary tree, the FPT algorithm removes 1 element and adds at

most 2 to the set \mathcal{Q} at each call. Hence, the cardinality $|\mathcal{Q}| \leq \ell$. Other lines of the FPT algorithm are some naive sequence manipulations with a computational time of $\mathcal{O}(N)$. Therefore, the total time complexity¹⁸ of the FPT algorithm is

$$T_{\text{avg}}^{\text{FPT}} = \underbrace{\mathcal{O}\left(\mathbb{E}_{\ell}\left[\sum_{j=1}^{\ell} \log j\right]\right)}_{\text{priority queue}} + \underbrace{\mathcal{O}(\ell_{\text{avg}} N)}_{\text{sequence manipulation}} = \mathcal{O}(\ell_{\text{avg}}(\log \ell_{\text{max}} + N)), \quad (133)$$

where the first term is simplified by

$$\mathbb{E}_{\ell}\left[\sum_{j=1}^{\ell} \log j\right] \leq \mathbb{E}_{\ell}\left[\sum_{j=1}^{\ell} \log \ell_{\text{max}}\right] = \mathbb{E}_{\ell}[\ell \log \ell_{\text{max}}] = \ell_{\text{avg}} \log \ell_{\text{max}}. \quad (134)$$

2) *Two-way FPT*: The FPT algorithm presented above is aimed at generating an un-constrained sequence over \mathbb{F}_2^N . We can use the FPT under constraints using the BMA [16] algorithm.

The two-way FPT (tFPT), like the SLVA, can generate ℓ best codewords of a binary linear code $\mathcal{C}[N, K]$ with a parity-check matrix \mathbf{P} . The basic idea of the tFPT is to separate the TEP into two halves, denoted by part A and part B. Part A (part B) is searched by the FPT algorithm, generating a list, called list A (list B), of ordered but un-constrained candidates. Then, two parts of TEPs, one from list A and the other from list B, are merged into a valid TEP if and only if the sum of their (partial) syndromes matches the total syndrome. Finally, all valid TEPs will be sorted before output. In practice, the above procedure can be implemented in a “calculate-when-request” manner to reach a low complexity.

In the following example (Example 19), we use the tFPT as the LGA for the LC-OSD and measure the (simulated) throughput of LC-OSD at different SNRs.

Example 19. In Fig. 16, we present the performance of two CRC-aided polar (CA-polar) codes with different lengths, namely $\mathcal{C}_{\text{CA-polar}}[128, 64]$ and $\mathcal{C}_{\text{CA-polar}}[256, 192]$, both of which are with 11 bits CRCs. The simulated throughput of the LC-OSD and the SCL decoders are also presented, where both the LC-OSD and the SCL decoders [33] are implemented in C++. All of

¹⁸Because the priority queue can only store the soft weights and references to TEPs, its time cost is irrelevant to N .

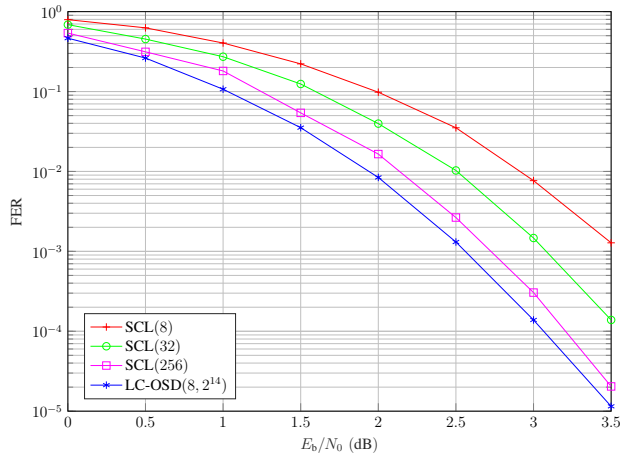
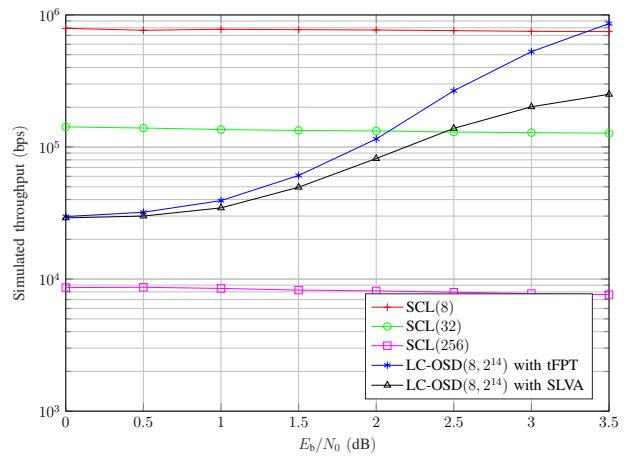
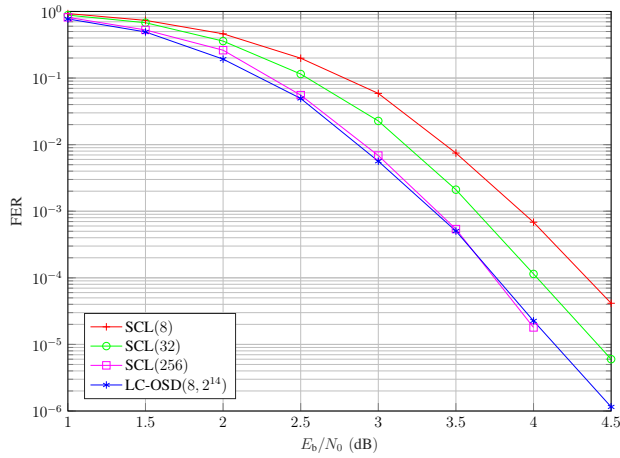
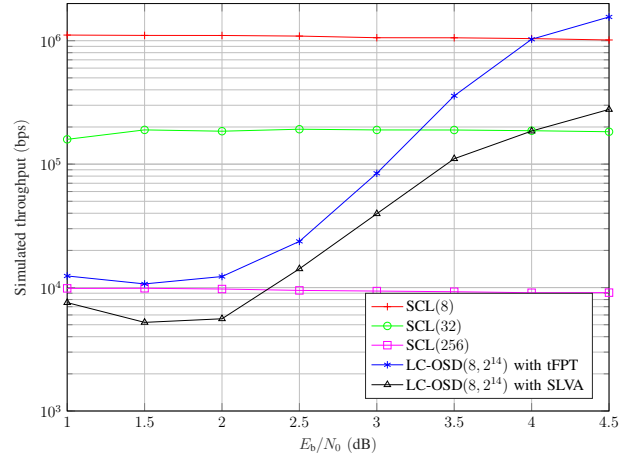
(a) Performance of the CA-polar code $\mathcal{C}_{\text{CA-polar}}[128, 64]$.(b) Throughput of the CA-polar code $\mathcal{C}_{\text{CA-polar}}[128, 64]$.(c) Performance of the CA-polar code $\mathcal{C}_{\text{CA-polar}}[256, 192]$.(d) Throughput of the CA-polar code $\mathcal{C}_{\text{CA-polar}}[256, 192]$.

Fig. 16. Simulation results of the CA-polar codes, comparing the LC-OSD with the SCL decoding [32].

the above decoders are run on the same computer with the same environment¹⁹. From Figs. 16b and 16d, we can see that, as the SNR increases, the throughputs of LC-OSDs (including the tFPT and SLVA versions) in the presented examples keep increasing, which is consistent with our predictions. In contrast, the throughput of a (particular) SCL decoder is relatively steady, no

¹⁹We have measured the throughput in a single thread, respectively, on a desktop computer with a CPU of 11th Gen Intel(R) Core(TM) i7-11700F @ 2.50GHz.

matter how high the SNR is. Comparing with the performance as well as the throughput, we conclude that the LC-OSD outperforms the successive cancellation list (SCL) decoding [32], especially in the high SNR region. From the figure, we also see that the LC-OSD with tFPT has a higher throughput than the SLVA in the high SNR region. This is due to the great portion of time consumed by the first candidate of the SLVA, which is also consistent with our predictions.

V. CONCLUSION

In this paper, we have presented an extended version of OSD called the OSD with local constraints (LC-OSD) algorithm. The LC-OSD uses more MRBs than the original OSD and solves the local constraint by list-generating algorithms (LGAs). The two presented LGAs, namely the SLVA and the tFPT, are suitable for different scenarios, where the SLVA is suitable for low SNR (typically requiring a significant amount of searches), and tFPT is for high SNR (typically requiring a small number of searches). We have also presented several early stopping criteria for the LC-OSD that significantly reduce computational complexity without noticeable performance loss. To analyze LC-OSD performance, we have proposed a method that uses random code to approximate the local constraint over MRBs. Several examples show that the approximation is quite accurate and does not depend on the structure of the codes. Based on the high accuracy of the approximation, we have proposed a tight approximate upper bound of the LC-OSD, which is applicable to different codes and decoding parameters. The random coding approach can lead to many research and applications related to the LC-OSD, one of which is tuning the decoding parameters according to applications.

ACKNOWLEDGMENT

The authors would like to express their gratitude to their colleague Dr. S Cai for his enlightening discussions and to Dr. M Fossorier for bringing [8] to their attention, of which they were

previously unaware.

REFERENCES

- [1] E. Berlekamp, R. McEliece, and H. Van Tilborg, “On the inherent intractability of certain coding problems (corresp.),” *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.
- [2] M. Fossorier and S. Lin, “Soft-decision decoding of linear block codes based on ordered statistics,” *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, 1995.
- [3] C. Yue, M. Shirvanimoghaddam, Y. Li, and B. Vucetic, “Segmentation-discarding ordered-statistic decoding for linear block codes,” in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
- [4] C. Yue, M. Shirvanimoghaddam, G. Park, O.-S. Park, B. Vucetic, and Y. Li, “Linear-equation ordered-statistics decoding,” *IEEE Transactions on Communications*, vol. 70, no. 11, pp. 7105–7123, 2022.
- [5] —, “Probability-based ordered-statistics decoding for short block codes,” *IEEE Communications Letters*, vol. 25, no. 6, pp. 1791–1795, 2021.
- [6] Y. Wang, J. Liang, and X. Ma, “Local constraint-based ordered statistics decoding for short block codes,” in *2022 IEEE Information Theory Workshop (ITW)*, 2022, pp. 107–112.
- [7] J. Liang, Y. Wang, S. Cai, and X. Ma, “A low-complexity ordered statistic decoding of short block codes,” *IEEE Communications Letters*, vol. 27, no. 2, pp. 400–403, 2023.
- [8] H. Greenberger, “An efficient soft decision decoding algorithm for block codes,” Jet Propulsion Laboratory, Pasadena, Calif., Tech. Rep., January and February 1979.
- [9] A. May, A. Meurer, and E. Thomae, “Decoding random linear codes in $O(2^{0.054n})$,” in *Advances in Cryptology—ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings 17*. Springer, 2011, pp. 107–124.
- [10] A. Becker, A. Joux, A. May, and A. Meurer, “Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding,” in *Advances in Cryptology—EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*. Springer, 2012, pp. 520–536.
- [11] A. May and I. Ozerov, “On computing nearest neighbors with applications to decoding of binary linear codes,” in *Advances in Cryptology—EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Springer, 2015, pp. 203–228.
- [12] A. Martinez and A. G. i Fàbregas, “Saddlepoint approximation of random-coding bounds,” in *2011 Information Theory and Applications Workshop*, 2011, pp. 1–6.
- [13] J. Font-Segura, G. Vazquez-Vilar, A. Martinez, A. Guillén i Fàbregas, and A. Lancho, “Saddlepoint approximations of lower and upper bounds to the error probability in channel coding,” in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, 2018, pp. 1–6.
- [14] N. Seshadri and C.-E. Sundberg, “List Viterbi decoding algorithms with applications,” *IEEE Transactions on Communications*, vol. 42, no. 234, pp. 313–323, 1994.
- [15] C. Nill and C.-E. Sundberg, “List and soft symbol output Viterbi algorithms: extensions and comparisons,” *IEEE Transactions on Communications*, vol. 43, no. 2/3/4, pp. 277–287, 1995.
- [16] A. Valembois and M. Fossorier, “Box and match techniques applied to soft-decision decoding,” *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 796–810, 2004.
- [17] C. Yue, M. Shirvanimoghaddam, B. Vucetic, and Y. Li, “A revisit to ordered statistics decoding: Distance distribution and decoding rules,” *IEEE Transactions on Information Theory*, vol. 67, no. 7, pp. 4288–4337, 2021.
- [18] A. Valembois and M. Fossorier, “A comparison between “most-reliable-basis reprocessing” strategies,” *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 85, no. 7, pp. 1727–1741, 2002.
- [19] M. Baldi, N. Maturo, E. Paolini, and F. Chiaraluce, “On the use of ordered statistics decoders for low-density parity-check codes in space telecommand links,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, pp. 1–15, 2016.
- [20] S. Tang, S. Cai, and X. Ma, “A new Chase-type soft-decision decoding algorithm for Reed-Solomon codes,” *Alexandria Engineering Journal*, vol. 61, no. 12, pp. 13 067–13 077, 2022.
- [21] X. Ma and A. Kavcic, “Path partitions and forward-only trellis algorithms,” *IEEE Transactions on Information Theory*, vol. 49, no. 1, pp. 38–52, 2003.

- [22] G. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.
- [23] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.
- [24] S. Lin, T. Kasami, T. Fujiwara, and M. Fossorier, *Trellises and trellis-based decoding algorithms for linear block codes*. Springer Science & Business Media, 1998.
- [25] W. Lin, "Research on block Markov superposition transmission for ultra-reliable and low-latency communication," Master's thesis, Sun Yat-sen University, 2020.
- [26] G. He, J.-C. Belfiore, I. Land, G. Yang, X. Liu, Y. Chen, R. Li, J. Wang, Y. Ge, R. Zhang, and W. Tong, "Beta-expansion: A theoretical framework for fast and recursive construction of polar codes," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.
- [27] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "Construction of polar codes with sublinear complexity," *IEEE Transactions on Information Theory*, vol. 65, no. 5, pp. 2782–2791, 2019.
- [28] D. E. Knuth, *The art of computer programming. Vol. 1: Fundamental algorithms*. Addison-Wesley, 1978.
- [29] S. Tang, S. Cai, and X. M. Member, "A new Chase-type soft-decision decoding algorithm for Reed-Solomon codes," *arXiv preprint arXiv:1309.1555*, 2013.
- [30] Z. Chen, S. Cai, L. Chen, and X. Ma, "Semi-LDPC convolutional codes with low-latency decoding algorithm," in *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, 2020, pp. 105–109.
- [31] A. Solomon, K. R. Duffy, and M. Médard, "Soft maximum likelihood decoding using GRAND," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.
- [32] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.
- [33] A. Cassagne, O. Hartmann, M. Leonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly *et al.*, "AFF3CT: A fast forward error correction toolbox!" *SoftwareX*, vol. 10, p. 100345, 2019.