Article

Pre-configured Error Pattern Ordered Statistics Decoding for CRC-Polar Codes

Xuanyu Li , Kai Niu [*] , Yuxin Han , Jincheng Dai , Zhiyuan Tan , Zhiheng Guo

Article

Pre-Configured Error Pattern Ordered Statistics Decoding for CRC-Polar Codes

Xuanyu Li [1] , **Kai Niu** [1,*] , **Yuxin Han** [1] , **Jincheng Dai** [1] , **Zhiyuan Tan** [2] **and Zhiheng Guo** [2]

[1]   Key Laboratory of Universal Wireless Communications, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing, China; {lixuanyu, niukai, hanyx, daijincheng}@bupt.edu.cn

[2]   Huawei Technologies. Co., Ltd, Shenzhen, Guangdong, China.; {tanzhiyuan, guozhiheng}@huawei.com

*   Correspondence: niukai@bupt.edu.cn

Abstract: In this paper, we propose a pre-configured error pattern ordered statistics decoding (PEPOSD) algorithm and discuss its application to short cyclic redundancy check (CRC)-polar codes. Unlike the traditional OSD that changes the most reliable independent symbols, we regard the decoding process as testing the error patterns, like guessing random additive noise decoding (GRAND). Also, the pre-configurator referred from ordered reliability bits (ORB) GRAND can better control the range and testing order of EPs. Offline-online structure can accelerate the decoding process. Additionally, we also introduce two orders to optimize the search order for testing EPs. Compared with CRC-aided OSD and list decoding, PEPOSD can achieve a better trade-off between accuracy and complexity.

Keywords: OSD; GRAND; channel decoding; polar code

---

1. Introduction

In ultra-reliable and low latency communications (URLLC), the high reliability of short block codes becomes the key requirement [1]. To do this, cyclic redundancy check (CRC-polar codes are particularly effective [2]. For decoding short CRC-polar codes, the state-of-the-art method is CRC-Aided (CA) - successive cancellation list (SCL) decoding [3].

Two cutting-edge short code decoding algorithms are ordered statistics decoding (OSD) [4] and guessing random additive noise decoding (GRAND) [5]. OSD is a decoder near the maximum likelihood (ML) and ideal for parallel design. However, the decoding complexity of s-order OSD can be too high to address.

Therefore, many pieces of early research have been done to reduce the complexity of OSD [6–9]. Recently, a threshold-based OSD decoder can reduce the number of tested codewords [10]. CA-OSD [10] and segmentation-discarding decoding [11] limit the number of valid codewords to improve performance. Probability-based OSD [12] calculates the promising probability and success probability to discard the candidate codewords.

Moreover, on the other hand, GRAND provides a new perspective for ML decoding by estimating the noise sequence [5]. Ordered reliability bits GRAND (ORBGRAND) [13] is proposed to improve decoding throughput by generating possible error patterns (EPs). Its high-throughput and energy-efficient very large-scale integration (VLSI) circuit architecture is given in [14].

In this paper, we propose a new scheme called pre-configured error pattern (PEP) OSD that considers OSD from a new perspective. The main innovations and the advantages of this scheme are summarized as follows:

(1) Decoding process: Instead of concentrating on completing queries of the most reliable independent symbols [4] on Hamming balls as s-order OSD, we use plenty of pre-configured EPs like ORBGRAND onto the transformed information bits. Before decoding, massive EPs can be pre-configured, so the EPs can be continuously read and tested on the hard-decision bits to see if these EPs can fix the errors in the information bits of the permuted systematic polar codes. After a Euclidean distance competition of δ codewords that can pass the CRC check, the most possible result can be

obtained. Due to the characteristics of CRC-polar codes, introducing the maximum number of valid codewords δ can early stop the decoding to achieve lower complexity.

(2) EP pre-configuring process: The EPs can be either pre-configured once for all kinds of codes (with different lengths or rates) to achieve higher decoding speed or dynamically generated before decoding to save the hardware resource. As optimizing the test order of the pre-configured EPs can further leverage the soft information, queries can be obviously saved. Two orders are introduced: index weight (IW) & Hamming weight (HW) order and priority weight (PW) order. IW&HW relates to the error possibility of a specific EP, and IW similar to the logical weight in ORBGRAND [13], though only for the transformed information bits in this scheme. Thus the possible calculating complexity is reduced. Moreover, PW, in a quantitative relationship related to IW and HW, is designed to direct an efficient way to use the possible EPs.

The remainder of this work is structured as follows: Preliminaries are provided in Section 2. The design of a PEPOSD decoder is given in Section 3. The generating theory and mechanism of PEP and testing order are given in Section 4. The simulations are evaluated in Section 5. Finally, conclusions are drawn in Section 6.

2. Preliminaries

2.1. CRC-Polar Codes

A CRC-polar code is characterized by its code length n, k-length information bits, and m-length CRC, thus denoted by $[n, k + m]$. For CRC-polar codes, the information bits are assigned to the channels with indices in the information set $\mathcal{A}$, related to the more reliable subchannels, and $|\mathcal{A}| = k + m$. The frozen bits, which have the default values, all zeros, are assigned to the complementary set $\mathcal{A}^c$. The channel input depends on the encoding function

$$f : \mathbf{c} = \mathbf{u} \cdot \mathbf{G}_n, \tag{1}$$

where $\mathbf{u}$ and $\mathbf{c}$ are the source and code block, respectively. The source block $\mathbf{u}$ consists of information bits $\mathbf{u}_{\mathcal{A}}$ and frozen bits $\mathbf{u}_{\mathcal{A}^c}$, and then modulated into BPSK vector $\mathbf{x}$. Suppose that $\mathbf{x}$ is transmitted over a noisy channel, and the received vector $\mathbf{y}$ is represented as

$$\mathbf{y} = \mathbf{x} + \mathbf{z}, \tag{2}$$

where $\mathbf{z}$ is the additive Gaussian noise. Therefore, there is

$$\theta(\mathbf{y}) = \mathbf{x} \oplus \mathbf{e}, \tag{3}$$

where $\theta(\mathbf{y})$ denotes the hard decision sequence of the received vector, and $\mathbf{e}$ denotes the EP where the "1" bits result in the flips of bits between the sequence sent and the hard decision of the received.

Note that the i-th element of a vector is expressed by [ ], for example, the i-th bit of the code is denoted by $c[i]$.

2.2. OSD Algorithm

In OSD, two permutations λ_1, λ_2 are performed over $\mathbf{y}$ and $\mathbf{G}$ before decoding. After these, the received signals $\tilde{\mathbf{y}}$ and the hard decision $\tilde{\theta}(\mathbf{y})$ are all respectively reordered. For example, $\mathbf{y}$ is reordered by

$$\tilde{\mathbf{y}} = \lambda_2(\lambda_1(\mathbf{y})). \tag{4}$$

Meanwhile, the permutations and Gaussian elimination transform the generator matrix $\mathbf{G}$ into its systematic form $\widetilde{\mathbf{G}}$ [3]. Therefore, only the $k + m$ most reliable positions of $\tilde{\mathbf{y}}$ are considered.

Then a number of tested codewords are compared to find the most likely estimate. In traditional OSD, codeword estimates are tested in the increasing order of the EP's Hamming weights. For instance, in s-order OSD, codeword estimates with Hamming weight from 1 to s of the corresponding EP are compared. After performing inverse permutations, the best result of the codeword estimates is chosen as the output.

3. PEPOSD Decoder

In this section, we introduce the details of PEPOSD. The whole decoder that can generate and test the EPs in parallel and relative processes is shown in Figure 1. There are two key units in PEPOSD: the offline pre-configurator and the online EP estimator. The pre-configurator can generate and reorder all the EPs and only once for all codes. The related details are described in Section 6. Meanwhile, the EP estimator consists of 3 modules: pre-processor, EP tester, and validity checker.
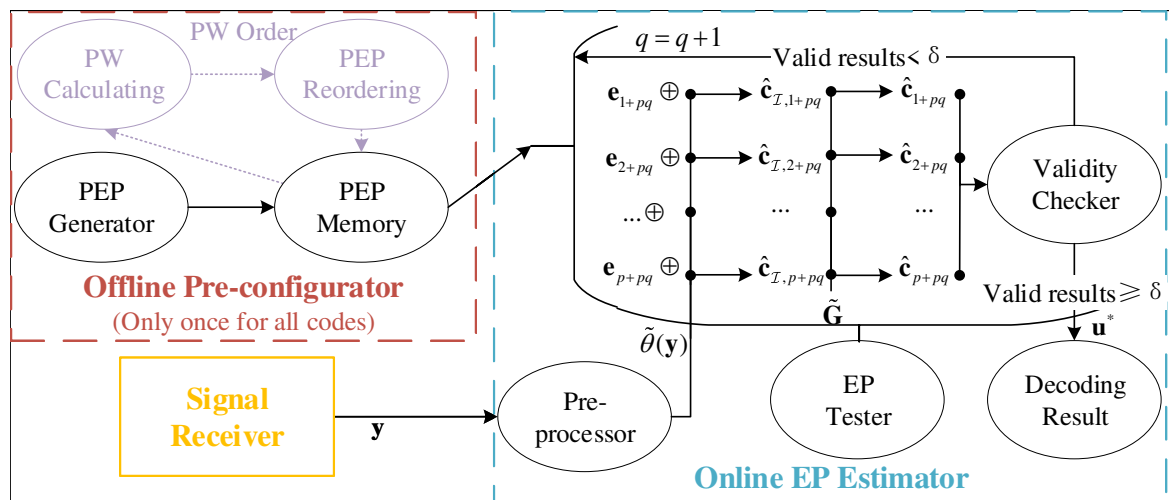

Figure 1. The offline-online structure of a PEPOSD decoder.

We also summarize the decoding process in Algorithm 1. Here we introduce the decoding process in detail.

Before decoding, the signals should be preprocessed by permutations λ_1 and λ_2. Thereby, the hard decision of the signal with $k + m$ systematic bits can be obtained. The EP tester then tests one or several EPs in parallel on the processed sequences and attains the possible result. The validity checker would decide if the result can pass the CRC check. The valid results will be stored in the list until the number reaches its limit δ. Otherwise, backtrack and another EP will be adopted and tested. Finally, the Euclidean distances of the δ results will be compared, and the most possible result will be selected as the decoding output.

The pre-processor performs two permutations and the systematic transform. The first permutation λ_1 sorts $\mathbf{y}$ by its absolute value $|\mathbf{y}|$, and the second permutation finds $k + m$ linearly independent column vectors in $\mathbf{G}$ as the first $k + m$ columns. Then it performs Gaussian elimination (GE) to the permuted generator matrix $\lambda_2(\lambda_1(\mathbf{G}))$, so the systematic form of generator matrix $\tilde{\mathbf{G}}$ is obtained. Thus the generator matrix becomes $\tilde{\mathbf{G}} = [\mathbf{I}_{k+m}, \tilde{\mathbf{P}}]$, where $\mathbf{I}_{k+m}$ is a $(k + m)$-dimensional identity matrix and $\tilde{\mathbf{P}}$ is the parity sub-matrix.

Meanwhile, perform λ_1 and λ_2 on the hard decision $\theta(\mathbf{y})$ and initial index $\mathbf{r}_0$, where $\mathbf{r}_0$ is set by $r_0[i] = i$. Then the reliability index $\mathbf{r}$ is obtained by $\lambda_2(\lambda_1(\mathbf{r}_0))$, which corresponds to the ascending-order index of reliability in the most reliable $k + m$ bits. The reordered-form $\tilde{\theta}(\mathbf{y})$, $\mathbf{r}$ can be obtained. Note that $\tilde{\theta}(\mathbf{y})$ consists of the first $(k + m)$ bits $\tilde{\theta}(\mathbf{y})_{\mathcal{I}}$ and the rest $\tilde{\theta}(\mathbf{y})_{\mathcal{P}}$, respectively corresponding to $\mathbf{I}_{k+m}$ and $\tilde{\mathbf{P}}$ in $\tilde{\mathbf{G}}$, i.e., $\tilde{\theta}(\mathbf{y}) = [\tilde{\theta}(\mathbf{y})_{\mathcal{I}}, \tilde{\theta}(\mathbf{y})_{\mathcal{P}}]$, where $\mathcal{I}$ and $\mathcal{P}$ denote the index set of the information and parity bits respectively.

Algorithm 1: PEPOSD for CRC-polar codes

Input: $N, \mathbf{y}, \mathbf{G}, w_{H,max}, w_{I,max}, \delta$

Output: $\mathbf{u}^*$

1   Read EPs from the memory

2   $\lambda_1 \leftarrow f : \text{Sort}(|\mathbf{y}|), \lambda_2 \leftarrow g : \text{Find\_Independent}(\mathbf{G})$

3   $\tilde{\mathbf{G}} = [\mathbf{I}_{k+m}, \tilde{\mathbf{P}}] \leftarrow \text{GE}(\lambda_2(\lambda_1(\mathbf{G}))), \tilde{\mathbf{r}} \leftarrow \lambda_2(\lambda_1(\mathbf{r}_0))$

4   $[\tilde{\theta}(\mathbf{y})_{\mathcal{I}}, \tilde{\theta}(\mathbf{y})_{\mathcal{P}}] = \tilde{\theta}(\mathbf{y}) \leftarrow \lambda_2(\lambda_1(\theta(\mathbf{y})))$

5   **for** $h = 1 : w_{C,max}$ **do**

6      $n \leftarrow \text{NumberofEP}(w_C = h)$ **for** $l = 1 : n$ **do**

7         $\mathbf{e} \leftarrow \mathbf{e}_l, \hat{\mathbf{c}}_{\mathcal{I}} \leftarrow \tilde{\theta}(\mathbf{y})_{\mathcal{I}} \ominus \mathbf{e}$

8         $\hat{\mathbf{c}} = [\hat{\mathbf{c}}_{\mathcal{I}}, \hat{\mathbf{c}}_{\mathcal{P}}] \leftarrow \hat{\mathbf{c}}_{\mathcal{I}} \cdot \tilde{\mathbf{G}}, \hat{\mathbf{c}} \leftarrow \lambda_1^{-1}(\lambda_2^{-1}(\hat{\mathbf{c}}))$

9         $\hat{\mathbf{u}} \leftarrow \hat{\mathbf{c}} \cdot \mathbf{G}$

10   **if** $\text{CRC\_Check}(\hat{\mathbf{u}}) = \text{true}$ **then**

11           $t \leftarrow t + 1, d^E \leftarrow \|\mathbf{y} - (1 - 2\hat{\mathbf{c}})\|^2$

12   **if** $d^E < d^E_{min}$ **then**

13             $\mathbf{u}^* \leftarrow \hat{\mathbf{u}}, d^E_{min} \leftarrow d^E$ **if** $t = \delta$ **then**

14   **return** $\mathbf{u}^*$

15   **end**

16   **end**

17   **end**

18   **end**

19 **end**

For each EP, the estimate of $\mathbf{x}$, is denoted by a codeword $\hat{\mathbf{c}}$. The systematic bits $\hat{\mathbf{c}}_I$ are generated by eliminating the error of hard decision $\tilde{\theta}(\mathbf{y})_{\mathcal{I}}$,

$$\hat{\mathbf{c}}_{\mathcal{I}} = \tilde{\theta}(\mathbf{y})_{\mathcal{I}} \oplus \mathbf{e}_l, \tag{5}$$

where $\mathbf{e}_l$ denotes the l-th EP. Then the whole codeword estimate $\hat{\mathbf{c}}$ can be calculated by

$$\hat{\mathbf{c}} = \hat{\mathbf{c}}_I \cdot \tilde{\mathbf{G}} = [\hat{\mathbf{c}}_{\mathcal{I}}, \hat{\mathbf{c}}_{\mathcal{P}}] = [\hat{\mathbf{c}}_{\mathcal{I}}, \hat{\mathbf{c}}_{\mathcal{I}} \cdot \tilde{\mathbf{P}}]. \tag{6}$$

Therefore, a possible candidate source block $\hat{\mathbf{u}}$ can be attained. After this, the validity checker will test if $\hat{\mathbf{u}}$ can pass the CRC check. If the CRC check is passed, $\hat{\mathbf{u}}$ is determined as a valid result and sent to the candidate list. Calculate the Euclidean distance $d^E = \|\mathbf{y} - (1 - 2\hat{\mathbf{c}})\|^2$ and compare it with the current minimum candidate d^E_{min}. If the number of candidates reaches δ, the decoding will be completed and the most likely candidate $\mathbf{u}^*$ will be output. This leverages the characteristic of CRC-polar codes to control the complexity.

If the candidate is invalid or the number is not enough, come back to the EP tester and read another EP. Though the generator matrix of CRC can be calculated into the whole generator matrix, a separate check is beneficial to control the number of queries.

4. Pre-Configured Error Patterns

In this section, we first discuss in IW&HW order, the theoretical basis of the PEP generating mechanism. Then two integer splitting algorithms are introduced. Finally, PW order is introduced to better control the testing order of the EPs.

4.1. IW&HW Order

As the reliability index $\mathbf{r}$ is obtained by $\lambda_2(\lambda_1(\mathbf{r}_0))$, this indicates the necessary order to eliminate the errors on these bits. Upon this, referring to ORBGRAND [13], we can define reliability weight

(RW), IW, and HW. The reliability weight is the sum of the approximate reliability of **e**, which can be calculated by

$$w_R(\mathbf{e}) = \sum_{i=1}^{k+m} \tilde{y}[i] \cdot e[i], \tag{7}$$

RW collects the reliability prior information of all permuted systematic bits. However, as RW is difficult to split and control, IW is introduced. For an error pattern **e**, the corresponding IW is defined as

$$w_I(\mathbf{e}) = \sum_{i=1}^{k+m} r[i] \cdot e[i], \tag{8}$$

which means the accumulation of the reliability index of all the error bits $e[i]$ given the specific EP **e**. The smaller IW generally corresponds to the bigger RW, and also the more possible noise effect of the specific EP. IW gives a quantitative integer indicator to evaluate the order to test EPs. The difference between IW and logical weight [13] is that IW only consists of the information of the systematic bits, which is determined endogenously by the OSD algorithm, and accordingly leads to different impacts. Furthermore, $w_{I,max}$ indicates the maximum IW in all the EPs.

Similarly, the HW of a given error pattern is defined as

$$w_H(\mathbf{e}) = \sum_{i=1}^{k+m} e[i]. \tag{9}$$

$w_{H,max}$ presents the maximum HW of all the EPs. The smaller HW often leads to some more usual errors. Without ambiguity, for all eligible **e**, $w_I(\mathbf{e}), w_H(\mathbf{e})$ are abbreviated as w_I, w_H.

To pre-configure the EPs with all IW and HW we set, the process of PEP generation is designed as follows. We first generate EPs whose $w_H = 1$. While generating "new" EPs whose w_H is from 2 to $w_{H,max}$, the generator first reads the "old" EPs whose $w_H^* = w_H - 1$, storing into $\mathbf{E}_{old}$. By splitting only the biggest integer in old EPs and putting the small integers aside, corresponding new EPs can be generated. The algorithm is summarized in Algorithm 2. While splitting the integer, b stands for the biggest number and $a_1, a_2,...,$ and b are in ascending order. Thus, all EPs needed can be pre-configured. An integer-splitting algorithm for ORBGRAND [13] can also be referred to.

Algorithm 2: Generate PEPs

Input: $w_{H,max}, w_{I,max}$

1 **for** $w_I = 1 : w_{I,max}$ **do**
2   $e_1[r[w_I]] \leftarrow 1$ // Gen. $w_H = 1$ EP
3   **for** $w_H = 2 : w_{H,max}$ **do**
4     $\mathbf{E}_{old} = \text{ReadPEP}(w_H - 1)$
5   **for** $i = 1 : Row(\mathbf{E}_{old}[i])$ **do**
6       $\{a_1, a_2, ..., a_{w_H-2}, b\} = \mathbf{E}_{old}[i]$
        // a_1, a_2,..., and b are in ascending order
7   **for** $a_{w_H-1} = max\{1, a_{w_H-2}\} : \lfloor (b-1)/2 \rfloor$ **do**
          // split b into 2 numbers
8         $\mathbf{E}_{new} = \mathbf{E}_{new} \cup \{a_1, a_2, ..., a_{w_H-2}, a_{w_H-1}, b - a_{w_H-1}\}$
9   **end**
10   **end**
11 **end**
12 **end**

There is an example for $w_{H,max} = 4$ and $w_I = 10$ shown in Figure 2. First the EP with $w_H = 1$ is generated. Then 10 is divided into {9,1},···, {6,4}, and 4 EPs with $w_H = 2$ are obtained. After that, 9 in

{9,1} can be divided into {7,2}, {6,3} and {5,4}, while 8 in {8,2} can be divided into {5,3}, thus 4 EPs with $w_H = 3$ are obtained. Finally, one EP with $w_H = 4$ is generated by dividing 7 in {7,2,1} into {4,3}.

PEP pre-configurator can produce all EPs stored in the memory before decoding numerous codes, so the decoder can continuously read EPs to significantly reduce the decoding delay, and only once is enough for all kinds of codes and all code blocks. On the other hand, while decoding a small number of codes, each EP can also be dynamically generated just before being tested to ensure better energy efficiency.
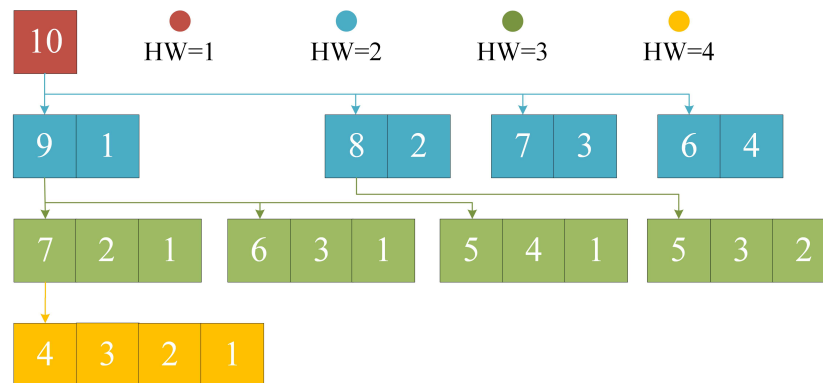

Figure 2. An example of generating for $w_{H,max} = 4$ and $w_I = 10$.

4.2. PW Order

As IW and HW are introduced and all the EPs have been pre-configured, PW can be defined by

$$w_P(\mathbf{e}) = w_I + \alpha \cdot (w_H)^\beta. \tag{10}$$

where α and β are parameters to be set. The order of using the EPs depends on their PW, which indicates a special order to prevent the decoder from trying some EPs with a super low possibility even if its HW is small.

Figure 3 gives a hypothetical example to see the difference of the PEPOSD scheme between IW&HW order and PW order. Figure 3(a) shows the IW&HW-order PEPOSD. The decoder first tests the $w_H = 1$ EPs in the order of w_I. After that, it tests those with $w_H = 2$, then 3, and so on. Meanwhile, in our new proposed scheme, the decoder just tests the EPs in the order of PW. Figure 3(b) shows how the PWs of the EPs correspond to their HW and PW. Therefore for instance, the EPs are tested from $w_P = 2$, to $w_P = 23$. Obviously, the two EPs with $w_P = 7$ are tested together, also for $w_P = 8, 9, 10, 15$. In this way, the order of using EPs can be optimized and some less probable EPs will be tested far later.
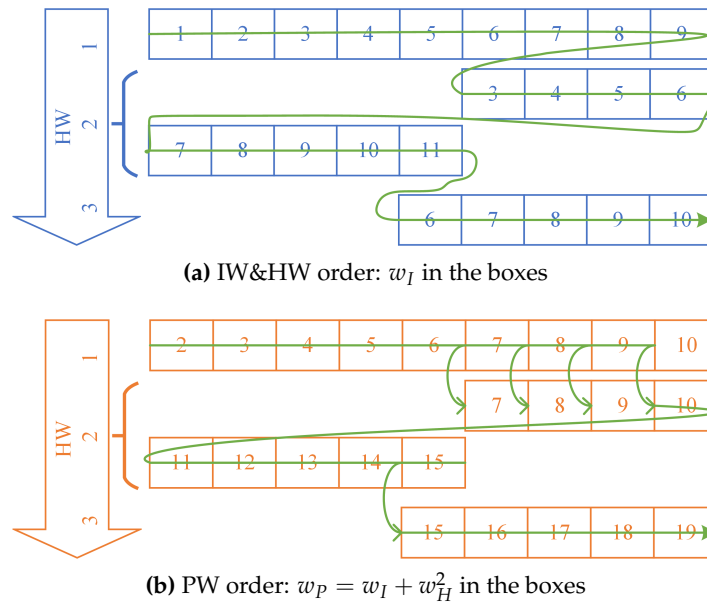
(a) IW&HW order: w_I in the boxes

(b) PW order: $w_P = w_I + w_H^2$ in the boxes

Figure 3. The sketches of IW&HW order and PW order.

5. Performance Evaluation

In this section, for CRC-polar codes, we respectively compare the performance of PEPOSD with 3-order CA-OSD and CA-SCL ($L = 32$).

5.1. BLER Analysis

First we compare the BLER performance with low complexity of these algorithms. Figure 4 shows the BLER comparison between PEPOSD (IW&HW) and CA-SCL with different rates with the code length $n = 64$ and CRC length $m = 6$. In this figure, there is $(\text{IW}/\text{HW}/\delta) = (75/4/20)$ for PEPOSD. This demonstrates that PEPOSD outperforms CA-SCL by about 0.3 dB with the close complexity for the high rates. Increasing the CRC length can improve the performance of PEPOSD while this worsens CA-SCL, so the advantage can be more obvious.
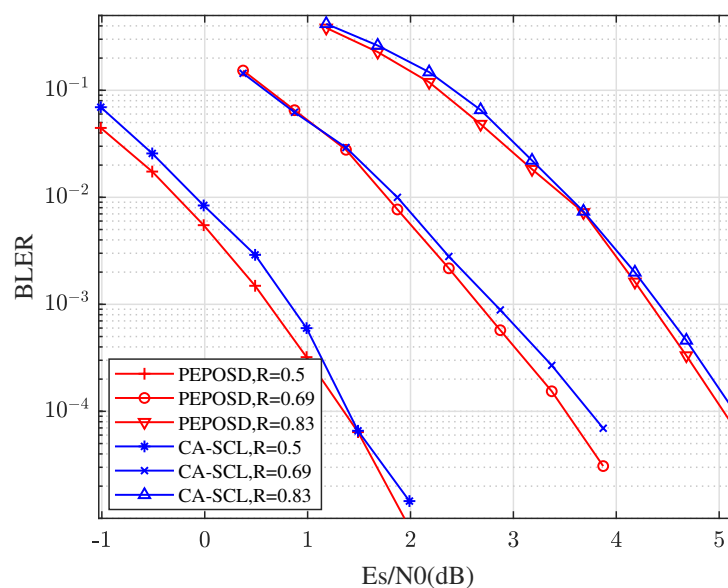

Figure 4. The comparison of BLER performance between PEPOSD and CA-SCL with different rates with the code length $n = 64$.

Figure 5 shows when $E_b/N_0 = 4.0$, the BLER comparison with different code rates from 0.5 to 0.85 among PEPOSD, CA-OSD, and CA-SCL. This shows that $PEPOSD_2$, $(75/4/20)$ can achieve close accuracy with CA-OSD. Moreover, when $R = 0.5$ and $R = 0.68$ or higher, PEPOSD outperforms CA-SCL obviously. More detailed analysis about PEPOSD related to its complexity is given in section 5.2.
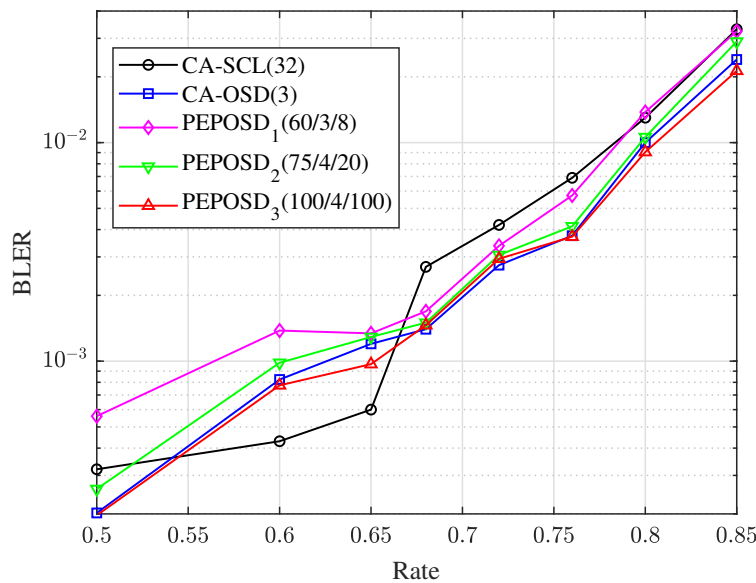

Figure 5. When SNR is 4.0dB, the BLER comparison with different code rate among PEPOSD, CA-OSD and CA-SCL.

Then we analyze the ultimate performance with higher complexity. Figure 6 shows the performance comparison for [128,108+11] CRC-polar code. PEPOSD(IW&HW) is here with $(IW/HW) = (100/4)$ and different δ. Meanwhile, PEPOSD(PW) with $(IW/HW/\delta/\alpha/\beta) = (100/4/1/2/3)$ and CA-SCL($L = 32$) are shown. The average decoding time is got from the same CPU. It can be concluded that PEPOSD can achieve better performance at a high rate for 128-bit CRC-polar codes and the decoding complexity can also be smaller than CA-SCL in high SNR areas. Also PW order performs better for this code.

Therefore, the simulation results show that PEPOSD achieves a better trade-off between accuracy and complexity than CA-OSD, and also can perform better for some short codes than CA-SCL. What's more, the parameters can be configured flexibly and the decoding process can be parallelized to further increase its throughput.
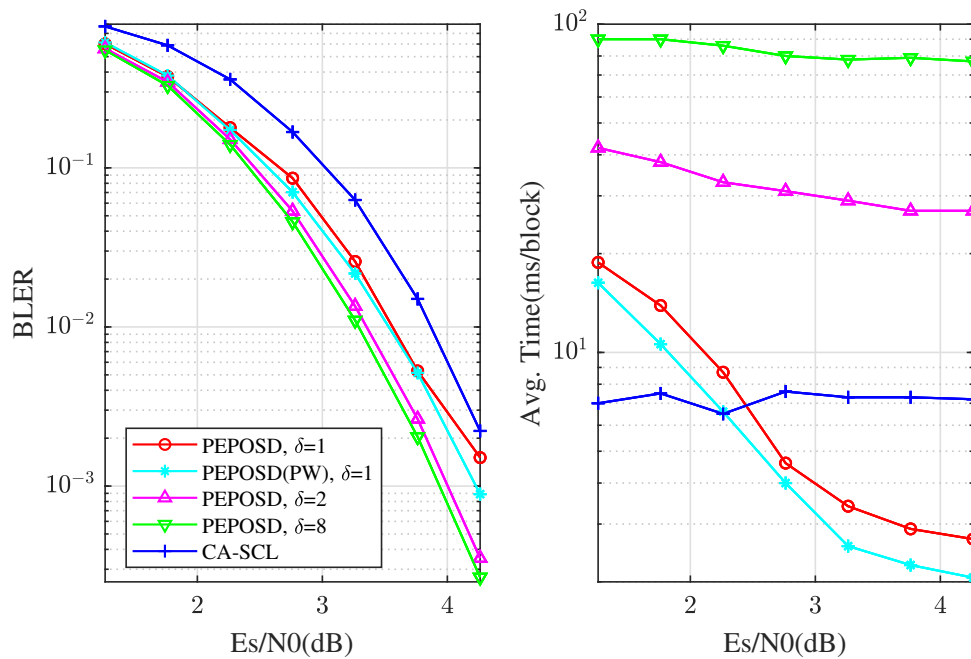
Figure 6. For [128,108+11] code, the comparison of BLER performance between PEPOSD and CA-SCL.

5.2. Complexity Analysis

First, we compare the computational complexity of the proposed scheme with CA-SCL. Specifically, it should be noted that all operations are modulo-two operations (XOR) in this scheme, so the hardware resources and time spent will be obviously less with the same quantity in the engineering practice and hardware implementations. As most of the OSD research does, we focus on the queries needed in the decoding process, also the number of the EPs tested. Therefore, the number of bit flipping in this period can be calculated by $\sum_{i=1}^{Q} w_H(e[i])$, where Q denotes the queries. Another key complexity that we consider compared with SCL, GRAND or other algorithms is GE in the pre-processor, of which the complexity can be calculated by $\mathcal{O}(n \cdot (min(k, n-k))^2)$. Moreover, there are some parallel or other efficient implementations can optimize the process like in [16].

Also, multiplication and addition operations needed in CA-SCL can be expressed as $\mathcal{O}(n \cdot L \cdot \log(n))$. Thus, Table 1 displays the complexity estimation of PEPOSD, CA-OSD, and CA-SCL. The queries of PEPOSD mainly based on δ, if IW and HW are relatively high enough. In conclusion, PEPOSD can obviously achieve lower complexity for high-rate codes, and for lower rates, PEPOSD may outperform CA-SCL as it's with modulo-two operations, which needs more hardware analysis to prove.

Table 1. Complexity Estimation of PEPOSD, CA-OSD and CA-SCL with different CRC-polar codes. For GE and CA-SCL, the number denotes the needed operations. For OSD algorithms, the number denotes the number of bit flipping.

Code	GE[*]	PEPOSD$_1$[**]	PEPOSD$_2$	PEPOSD$_3$	CA-OSD	CA-SCL
a.[64,32+6]	43264	898	2975	18535	8436	12288
b.[64,44+6]	12544	887	2722	18159	19600	12288
c.[64,53+6]	1600	899	2620	18108	32509	12288
d.[128,108+11]	28672	16	103	844	273819	10368

[*] Gaussian Eliminate is necessary in all OSD algorithms. [**] For PEPOSD$_1$,PEPOSD$_2$,PEPOSD$_3$: $n = 64, \delta = 8, 20, 100; n = 128, \delta = 1, 2, 8$.

Observing together with Figure 5, it's obvious that $PEPOSD_2, (75/4/20)$ can achieve close accuracy with CA-OSD while its average number of bit flipping is 1/9 to 1/36 of 3-order CA-OSD. Also, $PEPOSD_3, (100/4/100)$ can obtain better accuracy than CA-OSD and the queries can be greatly reduced at the same time. For high-rate codes, PEPOSD can outperforms CA-SCL and CA-OSD both in accuracy and complexity.

Finally, as PW is introduced, the queries reduction of the schemes with IW&HW and PW order is compared in Table 2. For [64,46+6] and [128,108+11] CRC-polar codes, using PEPOSD with $(IW/HW/\delta) = (100/4/1)$, the number of queries is reduced by 10%-30%.

Table 2. The average queries of IW&HW and PW order with $(IW/HW/\delta) = (100/4/1)$ for the CRC-polar codes.

Order	SNR=2.0dB[1]	SNR=2.5dB	SNR=3.0dB	SNR=3.5dB	SNR=4.0dB
(a) $n = 64$	$k = 46$	$m = 6$			
IW&HW	23.1	11	6.8	3.9	2.1
PW	16.5	11	5.9	3.5	2.1
(b) $n = 128$	$k = 108$	$m = 11$			
IW&HW	925	641	275	101	25
PW	928	514	188	75	22

[1] SNR denotes E_b/N_0.

6. Conclusion

In this paper, we introduce the PEPOSD algorithm to enhance the performance of short CRC-polar codes. It integrates the generating mechanism of noise queries in ORBGRAND to the generation of error patterns in OSD. Therefore, all the EPs can be pre-generated to allow the pipeline decoding for better speed. Also, early stop by CRC check can significantly reduce the complexity.

To optimize the decoding order of the proposed scheme, two options are introduced. IW&HW order is suitable for the most circumstances while PW shows lower complexity with bigger IW. In this way, the range of error patterns can be more controllable than l-order CA-OSD.

Simulation results show that there are several advantages in the performance and complexity of PEPOSD compared with CA-OSD and CA-SCL for CRC-polar codes, which shows a promising prospect.

Abbreviations

The following abbreviations are used in this manuscript:

PEPOSD	Pre-configured error pattern ordered statistics decoding
CRC	Cyclic redundancy check
GRAND	Guessing random additive noise decoding
ORB	Ordered reliability bits
URLLC	Ultra-reliable and low latency communications
CA	CRC-Aided
SCL	Successive cancellation list
ML	Maximum likelihood
EPs	Error patterns
VLSI	Very large-scale integration
IW	Index weight
HW	Hamming weight
PW	Priority weight

References

1. Liva G.; Gaudio L.; Ninacs T.; Jerkovits T. Code Design for Short Blocks: A Survey. arXiv:1610.00873, 2016. [CrossRef].
2. Niu K.; Zhang P.; Dai J.; Si Z.; Dong C. A golden decade of polar codes: From basic principle to 5G applications. *China Communications*, **2023**, 20, *94–121. [CrossRef].*
3. *Niu K.; Chen K. CRC-aided decoding of polar codes.* IEEE Commun. Lett., **2012**, 16, *1668–1671.[CrossRef].*
4. *Fossorier M. P. C.; Lin S. Soft-decision decoding of linear block codes based on ordered statistics.* IEEE Trans. Inf. Theory, **1995**, 41, *1379–1396. [CrossRef].*
5. *Duffy K. R.; Li J.; Médard M. Guessing noise, not code-words. In IEEE Int. Symp. Inf. Theory, 2018, pp. 671–675. [CrossRef].*
6. *Valembois A.; Fossorier M. An improved method to compute lists of binary vectors that optimize a given weight function with application to soft-decision decoding.* IEEE Commun. Lett., **2001**, 5, *456-458. [CrossRef].*
7. *Valembois A.; Fossorier. M. Box and match techniques applied to soft-decision decoding.* IEEE Trans. Inf. Theory, **2004**, 50, *796-810. [CrossRef].*
8. *Jin W.; Fossorier M. Efficient box and match algorithm for reliability-based soft decision decoding of linear block codes. In Proc. Inf. Theory Appl. Workshop, La Jolla, CA, USA, Jan. 2007. [CrossRef].*
9. *Wu Y.; Hadjicostis C. N. Soft-Decision Decoding Using Ordered Recodings on the Most Reliable Basis.* IEEE Trans. Inf. Theory, **2007**, 53, *829–836. [CrossRef].*
10. *Wu D.; Li Y.; Guo X.; Sun Y. Ordered Statistic Decoding for Short Polar Codes.* IEEE Commun. Lett., **2016**, 20, *1064-1067. [CrossRef].*
11. *Yue C.; Shirvanimoghaddam M.; Li Y.; Vucetic B. Segmentation-discarding ordered-statistic decoding for linear block codes. In Proc. IEEE Global Commun. Conf. (GLOBECOM), Waikoloa, HI, USA, 9-13 Dec. 2019, 1-6. [CrossRef].*
12. *Yue C.; Shirvanimoghaddam M.; Park G.; Park O.; Vucetic B.; Li Y. Probability-Based Ordered-Statistics Decoding for Short Block Codes.* IEEE Commun. Lett., **2021**, 25, *1791-1795. [CrossRef].*
13. *Duffy K. R.; An W.; Médard M. Ordered Reliability Bits Guessing Random Additive Noise Decoding.* IEEE Trans. Signal Process., **2022**, 70, *4528-4542. [CrossRef].*
14. *Abbas S. M.; Tonnellier T.; Ercan F.; Jalaleddine M. High-Throughput and Energy-Efficient VLSI Architecture for Ordered Reliability Bits GRAND.* IEEE Trans. on VLSI Systems, **2022**, 30. *[CrossRef].*
15. *Arıkan E. Channel polarization: a method for constructing capacity achieving codes for symmetric binary-input memoryless channels.* IEEE Trans. Inf. Theory, **2009**, 55, *3051–3073. [CrossRef].*
16. *Scholl S.; Stumm C.; Wehn N. Hardware implementations of Gaussian elimination over GF(2) for channel decoding algorithms. In 2013 Africon, Pointe aux Piments, Mauritius, Sept. 2013, 1-5. [CrossRef].*