

Adjacent-Bits-Swapped Polar codes: A new code construction to speed up polarization

Guodong Li

Min Ye

Sihuang Hu

Abstract

The construction of polar codes with code length $n = 2^m$ involves m layers of polar transforms. In this paper, we observe that after each layer of polar transforms, one can swap certain pairs of adjacent bits to accelerate the polarization process. More precisely, if the previous bit is more reliable than its next bit under the successive decoder, then switching the decoding order of these two adjacent bits will make the reliable bit even more reliable and the noisy bit even noisier.

Based on this observation, we propose a new family of codes called the Adjacent-Bits-Swapped (ABS) polar codes. We add a permutation layer after each polar transform layer in the construction of the ABS polar codes. In order to choose which pairs of adjacent bits to swap in the permutation layers, we rely on a new polar transform that combines two independent channels with 4-ary inputs. This new polar transform allows us to track the evolution of every pair of adjacent bits through different layers of polar transforms, and it also plays an essential role in the successive cancellation list (SCL) decoder for the ABS polar codes. Extensive simulation results show that ABS polar codes consistently outperform standard polar codes by 0.15 dB—0.3 dB when we use CRC-aided SCL decoder with list size 32 for both codes. The implementations of all the algorithms in this paper are available at <https://github.com/PlumJelly/ABS-Polar>

I. INTRODUCTION

Polar codes and Reed-Muller (RM) codes are two closely related code families in the sense that their generator matrices are formed of rows from the same square matrix. Although RM codes were discovered several decades earlier than polar codes, the capacity-achieving property of RM codes was established very recently. Specifically, polar codes were proposed by Arıkan in [2] and were shown to achieve capacity on all binary memoryless symmetric (BMS) channels in the same paper. In contrast, RM codes were proposed back in the 1950s [3], [4], but the question of whether RM codes achieve capacity remained open for more than 60 years until the recent breakthroughs. It was shown in [5] that RM codes achieve capacity on binary erasure channels (BEC) under the block-MAP decoder. More recently, Reeves and Pfister proved that RM codes achieve capacity on all BMS channels under the bit-MAP decoder [6].

While both code families achieve capacity of BMS channels, simulation results [7], [8] and theoretical analysis [9], [10] suggest that RM codes have better finite-length performance than polar codes. It was conjectured in [11] that this is because RM codes polarize even faster than polar codes. More precisely, in polar coding framework, we multiply a message vector consisting of $n = 2^m$ message bits with the matrix $\mathbf{G}_n^{\text{polar}} = (\mathbf{G}_2^{\text{polar}})^{\otimes m}$ and transmit the resulting codeword vector through a BMS channel. Here

Research partially funded by National Key R&D Program of China under Grant No. 2021YFA1001000, National Natural Science Foundation of China under Grant No. 12001322, Shandong Provincial Natural Science Foundation under Grant No. ZR202010220025, and a Taishan scholar program of Shandong Province. This paper was presented in part at the 2022 IEEE International Symposium on Information Theory [1].

Guodong Li is with School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, 266237, China. Email: guodongli@mail.sdu.edu.cn

Min Ye is with Tsinghua-Berkeley Shenzhen Institute, Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China. Email: yeemmi@gmail.com

Sihuang Hu is with Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, Shandong, 266237, China and School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, 266237, China. Email: husihuang@sdu.edu.cn

$\mathbf{G}_2^{\text{polar}} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, and \otimes is the Kronecker product. The message bits are divided into information bits and frozen bits according to their reliability under the successive decoder. This polar coding framework can also be used to analyze RM codes. To that end, we replace the recursive relation $\mathbf{G}_n^{\text{polar}} = \mathbf{G}_{n/2}^{\text{polar}} \otimes \mathbf{G}_2^{\text{polar}}$ in the standard polar code construction with $\mathbf{G}_n^{\text{RM}} = \mathbf{P}_n^{\text{RM}}(\mathbf{G}_{n/2}^{\text{RM}} \otimes \mathbf{G}_2^{\text{polar}})$. Here \mathbf{P}_n^{RM} is an $n \times n$ permutation matrix which reorders the rows of $\mathbf{G}_{n/2}^{\text{RM}} \otimes \mathbf{G}_2^{\text{polar}}$ according to their Hamming weights. It was conjectured in [11] that for the matrix \mathbf{G}_n^{RM} , the reliability of each message bit under the successive decoder becomes completely ordered, i.e., each message bit is always more reliable than its previous bit. If this conjecture were true, then one could show that RM codes polarize faster than polar codes, which leads to a better finite-length performance.

Inspired by the recursive relation $\mathbf{G}_n^{\text{RM}} = \mathbf{P}_n^{\text{RM}}(\mathbf{G}_{n/2}^{\text{RM}} \otimes \mathbf{G}_2^{\text{polar}})$ of RM codes, we propose a new family of codes called the Adjacent-Bits-Swapped (ABS) polar codes. In the construction of ABS polar codes, we use a similar recursive relation $\mathbf{G}_n^{\text{ABS}} = \mathbf{P}_n^{\text{ABS}}(\mathbf{G}_{n/2}^{\text{ABS}} \otimes \mathbf{G}_2^{\text{polar}})$. The matrix $\mathbf{P}_n^{\text{ABS}}$ is an $n \times n$ permutation matrix which swaps two adjacent rows if the two corresponding message bits are “unordered”, i.e., if the previous bit is more reliable than its next bit under the successive decoder. Swapping such two adjacent rows always accelerates polarization because it makes the reliable bit even more reliable and the noisy bit even noisier. While the permutation matrix \mathbf{P}_n^{RM} for RM codes involves a large number of swaps of adjacent rows, we limit the number of swaps in $\mathbf{P}_n^{\text{ABS}}$ so that the overall structure of ABS polar codes is still close to standard polar codes. In this way, we are able to devise a modified successive cancellation list (SCL) decoder to efficiently decode ABS polar codes.

Recall that both the code construction and the decoding algorithm of standard polar codes rely on a recursive relation between the bit-channels, which are the channels mapping from a message bit to the previous message bits and all the channel outputs. Since we swap certain pairs of adjacent bits in the ABS polar code construction, there is no explicit recursive relation between bit-channels. Instead, we introduce the notion of adjacent-bits-channels, which are 4-ary-input channels mapping from two adjacent message bits to the previous message bits and all the channel outputs. As the main technical contribution of this paper, we derive a recursive relation between the adjacent-bits-channels. This recursive relation serves as the foundation of efficient code construction and decoding algorithms for ABS polar codes.

We provide two sets of simulation results to compare the performance of ABS polar codes and standard polar codes. First, we empirically calculate the scaling exponents of ABS polar codes and standard polar codes over a binary erasure channel with erasure probability 0.5. Our calculations show that the scaling exponent of ABS polar codes is 3.37 while the scaling exponent of standard polar codes is 3.65, confirming that the polarization of ABS polar codes is indeed faster than standard polar codes. Second, we conduct extensive simulations over the binary-input AWGN channels for various choices of parameters. In particular, we have tested the performance for code length 256, 512, 1024, 2048. For each choice of code length, we test 3 code rates 0.3, 0.5 and 0.7. When we set the list size to be 32 for the CRC-aided SCL decoders of both code families, ABS polar codes consistently outperform standard polar codes by 0.15 dB—0.3 dB, but the decoding time of ABS polar decoder is longer than that of standard polar codes by roughly 60%. If we use list size 20 for ABS polar codes and keep the list size to be 32 for standard polar codes, then the decoding time is more or less the same for these two codes, and ABS polar codes still outperform standard polar codes for most choices of parameters. In this case, the improvement over standard polar codes is up to 0.15 dB.

The organization of this paper is as follows: In Section II, we describe the main idea behind the ABS polar code construction and explain why ABS polar codes polarize faster than standard polar codes. In Section III, we derive the new recursive relation between the adjacent-bits-channels and use this recursive relation to construct ABS polar codes. In Section IV, we present an efficient encoding algorithm for ABS polar codes. In Section V, we present the new SCL decoder for ABS polar codes. Finally, in Section VI, we provide the simulation results.

II. MAIN IDEA OF THE NEW CODE CONSTRUCTION

A. The polarization framework

Let U_1, U_2, \dots, U_n be n i.i.d. Bernoulli-1/2 random variables. Let \mathbf{G}_n be an $n \times n$ invertible matrix over the binary field. Define $(X_1, X_2, \dots, X_n) = (U_1, U_2, \dots, U_n)\mathbf{G}_n$. We transmit each X_i through a BMS channel W and denote the channel output vector as (Y_1, Y_2, \dots, Y_n) . In this framework, (U_1, U_2, \dots, U_n) is the message vector, \mathbf{G}_n is the encoding matrix, and (X_1, X_2, \dots, X_n) is the codeword vector. We use a successive decoder to recover the message vector from the channel output vector. More precisely, we decode the coordinates in the message vector one by one from U_1 to U_n . When decoding U_i , the successive decoder knows the values of all the previous message bits U_1, \dots, U_{i-1} and all the channel outputs Y_1, \dots, Y_n . Note that the codeword vector (X_1, X_2, \dots, X_n) depends on the matrix \mathbf{G}_n , and the channel output vector (Y_1, Y_2, \dots, Y_n) depends on both the matrix \mathbf{G}_n and the BMS channel W , although we omit the dependence from their notation. Next we define

$$H_i(\mathbf{G}_n, W) := H(U_i | U_1, \dots, U_{i-1}, Y_1, \dots, Y_n) \text{ for } 1 \leq i \leq n, \quad (1)$$

where $H(\cdot|\cdot)$ is the conditional entropy. $H_i(\mathbf{G}_n, W)$ measures the reliability of the i th message bit under the successive decoder when we use the encoding matrix \mathbf{G}_n and transmit the codeword vector through the BMS channel W . Since \mathbf{G}_n is an invertible matrix, we have

$$H_1(\mathbf{G}_n, W) + H_2(\mathbf{G}_n, W) + \dots + H_n(\mathbf{G}_n, W) = n(1 - I(W)), \quad (2)$$

where $I(W)$ is the channel capacity of W . We say that a family of matrices $\{\mathbf{G}_n\}$ is polarizing over a BMS channel W if $H_i(\mathbf{G}_n, W)$ is close to either 0 or 1 for almost all $i \in \{1, 2, \dots, n\}$ as $n \rightarrow \infty$. In order to quantify the polarization level of a given encoding matrix \mathbf{G}_n over a BMS channel W , we define

$$\Gamma(\mathbf{G}_n, W) = \frac{1}{n} \sum_{i=1}^n H_i(\mathbf{G}_n, W)(1 - H_i(\mathbf{G}_n, W)).$$

According to the definition above, a family of matrices $\{\mathbf{G}_n\}$ is polarizing over W if and only if $\Gamma(\mathbf{G}_n, W) \rightarrow 0$ as $n \rightarrow \infty$. A family of polarizing matrix $\{\mathbf{G}_n\}$ over a BMS channel W allows us to construct capacity-achieving codes as follows: We include the i th row of \mathbf{G}_n in the generator matrix if and only if $H_i(\mathbf{G}_n, W)$ is very close to 0. The condition $H_i(\mathbf{G}_n, W) \approx 0$ guarantees that the decoding error of the constructed codes approaches 0 under the successive decoder. We can further use (2) to show that the code rate of the constructed codes approaches $I(W)$. To see this, we first assume the extreme case where $\Gamma(\mathbf{G}_n, W) = 0$, i.e., $H_i(\mathbf{G}_n, W)$ is either 0 or 1 for all $1 \leq i \leq n$. Then by (2) we know that the dimension of the constructed polar code is precisely $nI(W)$, i.e., the code rate is $R = I(W)$. For the realistic case of $\Gamma(\mathbf{G}_n, W) \rightarrow 0$ as $n \rightarrow \infty$, one can show that the gap to capacity $I(W) - R$ also decreases to 0 as $n \rightarrow \infty$. Moreover, the smaller $\Gamma(\mathbf{G}_n, W)$ is, the smaller gap to capacity we have.

In the standard polar code construction [2], we construct the family of matrices $\{\mathbf{G}_{2^m}^{\text{polar}}\}_{m=1}^{\infty}$ recursively using the following relation:

$$\mathbf{G}_2^{\text{polar}} := \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \text{ and } \mathbf{G}_n^{\text{polar}} = \mathbf{G}_{n/2}^{\text{polar}} \otimes \mathbf{G}_2^{\text{polar}} \text{ for } n = 2^m \geq 4,$$

where \otimes is the Kronecker product and $m > 1$ is a positive integer. It was shown in [2] that $\{\mathbf{G}_{2^m}^{\text{polar}}\}_{m=1}^{\infty}$ is polarizing over every BMS channel W , and the codes constructed from these matrices can be efficiently decoded. In this paper, our objective is to construct another family of polarizing matrices $\{\mathbf{G}_{2^m}^{\text{ABS}}\}_{m=1}^{\infty}$ satisfying the following two conditions: (1) $\Gamma(\mathbf{G}_{2^m}^{\text{ABS}}, W) < \Gamma(\mathbf{G}_{2^m}^{\text{polar}}, W)$, i.e., the matrices $\mathbf{G}_{2^m}^{\text{ABS}}$ polarize even faster than $\mathbf{G}_{2^m}^{\text{polar}}$; (2) the codes constructed from $\{\mathbf{G}_{2^m}^{\text{ABS}}\}_{m=1}^{\infty}$ can also be efficiently decoded. The first condition allows us to construct a new family of codes with smaller gap to capacity and better finite-length performance than standard polar codes.

B. Swapping unordered adjacent bits accelerates polarization

The key observation in the standard polar code construction is that $\Gamma(\mathbf{G}_n, W)$ decreases as we perform the Kronecker product $\mathbf{G}_{2n} = \mathbf{G}_n \otimes \mathbf{G}_2^{\text{polar}}$. More precisely, we always have

$$\Gamma(\mathbf{G}_n \otimes \mathbf{G}_2^{\text{polar}}, W) < \Gamma(\mathbf{G}_n, W)$$

for every invertible matrix \mathbf{G}_n as long as $I(W)$ is not equal to 0 or 1. Therefore, the Kronecker product $\mathbf{G}_{2n} = \mathbf{G}_n \otimes \mathbf{G}_2^{\text{polar}}$ deepens the polarization at the cost of increasing the code length by a factor of 2.

In this paper, we observe that there is another method to deepen the polarization without increasing the code length, and this simple observation forms the foundation of our new code construction. Given a matrix \mathbf{G}_n and a BMS channel W , we say that two adjacent message bits U_i and U_{i+1} are unordered if $H_i(\mathbf{G}_n, W) \leq H_{i+1}(\mathbf{G}_n, W)$. This inequality means that U_i is more reliable than U_{i+1} under the successive decoder although U_i is decoded before U_{i+1} . Our key observation is that in this case, switching the decoding order of U_i and U_{i+1} deepens the polarization. Intuitively, this is because switching the decoding order of these two bits makes the reliable bit even more reliable and the noisy bit even noisier.

Note that switching the decoding order of U_i and U_{i+1} is equivalent to swapping the i th row and the $(i+1)$ th row of \mathbf{G}_n . More specifically, let us define a new matrix $\overline{\mathbf{G}}_n$ as the matrix obtained from swapping the i th row and the $(i+1)$ th row of \mathbf{G}_n and keeping all the other rows the same as \mathbf{G}_n . Following the framework in Section II-A, let $(\overline{U}_1, \overline{U}_2, \dots, \overline{U}_n)$ be the message vector associated with the new matrix $\overline{\mathbf{G}}_n$, where $\overline{U}_1, \dots, \overline{U}_n$ are n i.i.d. Bernoulli-1/2 random variables. Let $(\overline{X}_1, \dots, \overline{X}_n) = (\overline{U}_1, \dots, \overline{U}_n)\overline{\mathbf{G}}_n$ be the codeword vector transmitted through the BMS channel W and let $(\overline{Y}_1, \dots, \overline{Y}_n)$ be the corresponding channel output vector. By definition (1), we have

$$H_j(\overline{\mathbf{G}}_n, W) = H(\overline{U}_j | \overline{U}_1, \dots, \overline{U}_{j-1}, \overline{Y}_1, \dots, \overline{Y}_n) \text{ for } 1 \leq j \leq n.$$

By the relation between the matrices $\overline{\mathbf{G}}_n$ and \mathbf{G}_n , we have

$$H_j(\mathbf{G}_n, W) = H_j(\overline{\mathbf{G}}_n, W) \text{ for all } j \in \{1, 2, \dots, n\} \setminus \{i, i+1\}, \quad (3)$$

$$\begin{aligned} H_i(\mathbf{G}_n, W) &= H(\overline{U}_{i+1} | \overline{U}_1, \dots, \overline{U}_{i-1}, \overline{Y}_1, \dots, \overline{Y}_n) \\ &\geq H(\overline{U}_{i+1} | \overline{U}_1, \dots, \overline{U}_{i-1}, \overline{U}_i, \overline{Y}_1, \dots, \overline{Y}_n) = H_{i+1}(\overline{\mathbf{G}}_n, W), \end{aligned} \quad (4)$$

$$\begin{aligned} H_{i+1}(\mathbf{G}_n, W) &= H(\overline{U}_i | \overline{U}_1, \dots, \overline{U}_{i-1}, \overline{U}_{i+1}, \overline{Y}_1, \dots, \overline{Y}_n) \\ &\leq H(\overline{U}_i | \overline{U}_1, \dots, \overline{U}_{i-1}, \overline{Y}_1, \dots, \overline{Y}_n) = H_i(\overline{\mathbf{G}}_n, W). \end{aligned} \quad (5)$$

Now suppose that U_i and U_{i+1} are unordered, i.e., $H_i(\mathbf{G}_n, W) \leq H_{i+1}(\mathbf{G}_n, W)$. Combining this inequality with (4)–(5), we obtain

$$H_{i+1}(\overline{\mathbf{G}}_n, W) \leq H_i(\mathbf{G}_n, W) \leq H_{i+1}(\mathbf{G}_n, W) \leq H_i(\overline{\mathbf{G}}_n, W). \quad (6)$$

Moreover,

$$H_i(\mathbf{G}_n, W) + H_{i+1}(\mathbf{G}_n, W) = H_i(\overline{\mathbf{G}}_n, W) + H_{i+1}(\overline{\mathbf{G}}_n, W) = H(\overline{U}_i, \overline{U}_{i+1} | \overline{U}_1, \dots, \overline{U}_{i-1}, \overline{Y}_1, \dots, \overline{Y}_n).$$

This equality together with (6) implies that

$$\begin{aligned} &(H_i(\mathbf{G}_n, W))^2 + (H_{i+1}(\mathbf{G}_n, W))^2 \\ &= \frac{1}{2} \left((H_i(\mathbf{G}_n, W) + H_{i+1}(\mathbf{G}_n, W))^2 + (H_i(\mathbf{G}_n, W) - H_{i+1}(\mathbf{G}_n, W))^2 \right) \\ &\leq \frac{1}{2} \left((H_i(\overline{\mathbf{G}}_n, W) + H_{i+1}(\overline{\mathbf{G}}_n, W))^2 + (H_i(\overline{\mathbf{G}}_n, W) - H_{i+1}(\overline{\mathbf{G}}_n, W))^2 \right) \\ &= (H_i(\overline{\mathbf{G}}_n, W))^2 + (H_{i+1}(\overline{\mathbf{G}}_n, W))^2. \end{aligned}$$

Therefore,

$$H_i(\overline{\mathbf{G}}_n, W)(1 - H_i(\overline{\mathbf{G}}_n, W)) + H_{i+1}(\overline{\mathbf{G}}_n, W)(1 - H_{i+1}(\overline{\mathbf{G}}_n, W))$$

$$\leq H_i(\mathbf{G}_n, W)(1 - H_i(\mathbf{G}_n, W)) + H_{i+1}(\mathbf{G}_n, W)(1 - H_{i+1}(\mathbf{G}_n, W)).$$

Combining this with (3), we conclude that $\Gamma(\overline{\mathbf{G}}_n, W) \leq \Gamma(\mathbf{G}_n, W)$. This formally justifies that switching the decoding order of two unordered adjacent bits deepens polarization.

C. Our new code construction and its connection to RM codes

We view the operation of taking the Kronecker product $\mathbf{G}_n^{\text{polar}} = \mathbf{G}_{n/2}^{\text{polar}} \otimes \mathbf{G}_2^{\text{polar}}$ in the standard polar code construction as one layer of polar transform. Then the construction of a standard polar code with code length $n = 2^m$ consists of m consecutive layers of polar transforms. In light of the discussion in Section II-B, we add a permutation layer after each polar transform layer in our ABS polar code construction. More precisely, we replace the recursive relation $\mathbf{G}_n^{\text{polar}} = \mathbf{G}_{n/2}^{\text{polar}} \otimes \mathbf{G}_2^{\text{polar}}$ in the standard polar code construction with

$$\mathbf{G}_n^{\text{ABS}} = \mathbf{P}_n^{\text{ABS}}(\mathbf{G}_{n/2}^{\text{ABS}} \otimes \mathbf{G}_2^{\text{polar}}), \quad (7)$$

where the matrix $\mathbf{P}_n^{\text{ABS}}$ is an $n \times n$ permutation matrix. In this case, $\mathbf{G}_n^{\text{ABS}}$ is a row permutation of the Kronecker product $\mathbf{G}_{n/2}^{\text{ABS}} \otimes \mathbf{G}_2^{\text{polar}}$. The permutation associated with $\mathbf{P}_n^{\text{ABS}}$ is a composition of multiple swaps of unordered adjacent bits. The starting point of the recursive relation (7) is $\mathbf{G}_1^{\text{ABS}} = [1]$, the identity matrix of size 1×1 .

Before we present how to choose $\mathbf{P}_n^{\text{ABS}}$ in (7), let us point out an interesting connection between our new code and RM codes. In fact, RM codes can also be constructed using a similar recursive relation:

$$\mathbf{G}_n^{\text{RM}} = \mathbf{P}_n^{\text{RM}}(\mathbf{G}_{n/2}^{\text{RM}} \otimes \mathbf{G}_2^{\text{polar}}). \quad (8)$$

Here \mathbf{P}_n^{RM} is an $n \times n$ permutation matrix which reorders the rows of $\mathbf{G}_{n/2}^{\text{RM}} \otimes \mathbf{G}_2^{\text{polar}}$ according to their Hamming weights. In other words, \mathbf{G}_n^{RM} is a row permutation of $\mathbf{G}_{n/2}^{\text{RM}} \otimes \mathbf{G}_2^{\text{polar}}$, and the Hamming weights of the rows of \mathbf{G}_n^{RM} are monotonically increasing from the first row to the last row. It was shown in [11] that the family of matrices $\{\mathbf{G}_n^{\text{RM}}\}$ is polarizing over every BMS channel W , i.e., $H_i(\mathbf{G}_n^{\text{RM}}, W)$ is close to either 0 or 1 for almost all $i \in \{1, 2, \dots, n\}$ as $n \rightarrow \infty$. It was further conjectured¹ in [11] that $\{H_i(\mathbf{G}_n^{\text{RM}}, W)\}_{i=1}^n$ is decreasing for every BMS channel W , i.e.,

$$H_1(\mathbf{G}_n^{\text{RM}}, W) \geq H_2(\mathbf{G}_n^{\text{RM}}, W) \geq \dots \geq H_n(\mathbf{G}_n^{\text{RM}}, W). \quad (9)$$

If this conjecture were true, then we can immediately conclude that RM codes achieve capacity of BMS channels. Indeed, RM codes choose rows with heaviest Hamming weight in \mathbf{G}_n^{RM} to form the generator matrices. Since the rows of \mathbf{G}_n^{RM} are sorted according to their Hamming weights, RM codes simply pick the rows with large row indices. By (9), these rows correspond to the most reliable bits under the successive decoder. Moreover, since almost all the conditional entropy in (9) are close to either 0 or 1, the conditional entropy of the most reliable bits must be close to 0, and the number of such bits is close to $nI(W)$ as $n \rightarrow \infty$.

Moreover, the conjecture (9) indicates that RM codes do not have any unordered adjacent bits. According to the discussion in Section II-B, this suggests that RM codes have fast polarization. In fact, it is widely believed that RM codes have a smaller gap to capacity than polar codes with the same parameters, which was suggested to be the case by both theoretical analysis [9], [10] and simulation results [7], [8].

Although RM codes are believed to have better performance than polar codes under the Maximum Likelihood (ML) decoder, the problem of designing an efficient decoder whose performance is almost the same as the ML decoder still remains open for RM codes, except for a certain range of parameters. In particular, the performance of currently known decoding algorithms for RM codes [8], [12]–[14] is close to the ML decoder only in the short code length or the low code rate regimes. In contrast, the

¹The authors of [11] provided some theoretical analysis and simulation results to support this conjecture.

performance of the successive cancellation list (SCL) decoder with list size 32 is almost the same as the ML decoder for polar codes.

Our new code construction is an intermediate point between RM codes and polar codes. On the one hand, the recursive relation (7) of our new code is similar to the recursion (8) of RM codes in the sense that both codes add a permutation layer after each polar transform layer to accelerate polarization. On the other hand, we only use a relatively small number of swaps in the permutation matrix $\mathbf{P}_n^{\text{ABS}}$ while the permutation matrix \mathbf{P}_n^{RM} for RM codes involves a large number of swaps. As a consequence, the overall structure of our new code is still close to the standard polar codes, and it allows a modified SCL decoder to efficiently decode.

In order to explain how to choose $\mathbf{P}_n^{\text{ABS}}$ in (7), we introduce a sequence of permutation matrices. For $1 \leq i \leq n-1$, we use $\mathbf{S}_n^{(i)}$ to denote the $n \times n$ permutation matrix that swaps i and $i+1$ while mapping all the other elements to themselves. More precisely, only 4 entries of $\mathbf{S}_n^{(i)}$ are different from the identity matrix. These 4 entries are $\mathbf{S}_n^{(i)}(i, i) = \mathbf{S}_n^{(i)}(i+1, i+1) = 0$ and $\mathbf{S}_n^{(i)}(i, i+1) = \mathbf{S}_n^{(i)}(i+1, i) = 1$, where $\mathbf{S}_n^{(i)}(a, b)$ is the entry of $\mathbf{S}_n^{(i)}$ located at the cross of the a th row and the b th column. The permutation matrix $\mathbf{P}_n^{\text{ABS}}$ can be written as

$$\mathbf{P}_n^{\text{ABS}} = \prod_{i \in \mathcal{I}^{(n)}} \mathbf{S}_n^{(i)}, \quad (10)$$

where $\mathcal{I}^{(n)}$ is a subset of $\{1, 2, \dots, n-1\}$. Let us write $\mathcal{I}^{(n)} = \{i_1, i_2, \dots, i_s\}$, where s is the size of $\mathcal{I}^{(n)}$. In the ABS polar code construction, we require that

$$i_2 \geq i_1 + 4, \quad i_3 \geq i_2 + 4, \quad i_4 \geq i_3 + 4, \quad \dots, \quad i_s \geq i_{s-1} + 4. \quad (11)$$

This condition guarantees that the swapped elements are fully separated, and it is the foundation of efficient code construction and efficient decoding for ABS polar codes. More specifically, the condition (11) allows us to efficiently track the evolution of every pair of adjacent bits through different layers of polar transforms in a recursive way. We will explain the details about this in Section III. As a final remark, we note that one needs to choose m permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in the construction of an ABS polar code with code length $n = 2^m$.

D. Comparison with the large kernel method

The finite-length scaling of polar codes is an important research topic in the polar coding literature [9], [15]–[17]. The ABS polar code construction proposed in this paper is one way to improve the scaling exponent of polar codes. Another extensively-studied method is to use large kernels instead of the Arkan kernel $\mathbf{G}_2^{\text{polar}}$ in the polar code construction [18]–[23]. In particular, it was shown in [21]–[23] that when the kernel size goes to infinity, the scaling exponent of polar codes approaches the optimal value 2.

Compared to the ABS polar code construction, the large kernel method has the following three disadvantages: (i) The choice of code length is more restrictive. The code length of ABS polar codes can be any power of 2, but the code length of polar codes with large kernels must be a power of the kernel size ℓ , where ℓ is larger than 2. Some typical choices of ℓ are 4, 8, 16. (ii) The code construction is also more restrictive. In the original large kernel method, the same kernel is used repetitively throughout the whole code construction. In contrast, we use different permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in different layers. (iii) The decoding complexity is much larger. For $\ell \times \ell$ kernels, the decoding time increases by a factor of 2^ℓ compared to standard polar codes. In contrast, the decoding time of ABS polar codes only increases by 60% compared to standard polar codes, as indicated by the simulation results in Section VI.

Among the research on polar codes with large kernels, the permuted kernels and the permuted successive cancellation (PSC) decoder proposed in [18], [19] are particularly relevant to our paper. More specifically, [18], [19] proposed to use permuted kernels, whose size ℓ is a power of 2. As suggested by its name, the permuted kernel is a row permutation of $\mathbf{G}_\ell^{\text{polar}}$. This is similar in nature to the ABS polar code construction because the encoding matrix $\mathbf{G}_n^{\text{ABS}}$ of ABS polar codes is also a row permutation of

$\mathbf{G}_n^{\text{polar}}$. Moreover, [18], [19] further proposed the PSC decoder to efficiently decode polar codes with permuted kernels. The PSC decoder together with the permuted kernels significantly reduces the decoding time compared to the standard SC decoder for polar codes with large kernels. In other words, the PSC decoder and permuted kernels mitigate the third disadvantage above. However, the first two disadvantages still remain, i.e., the choice of code length and the code construction are still more restrictive than ABS polar codes.

III. CODE CONSTRUCTION OF ABS POLAR CODES

The construction of ABS polar codes with code length $n = 2^m$ consists of two main steps. The first step is to pick the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in the recursive relation (7), as mentioned at the end of the previous section. After picking these permutation matrices, the second step is to find which bits are information bits and which bits are frozen bits. Although the second step is also needed in the construction of standard polar codes [2], [24], the techniques used in this paper are quite different. In the standard polar code construction, we can directly track the evolution of bit-channels in a recursive way. However, in the ABS polar code construction, it is not possible to identify a recursive relation between bit-channels directly because we swap certain pairs of adjacent bits in the code construction. Instead, we find a recursive relation between pairs of adjacent bits from different layers of polar transforms. After obtaining the joint distribution of every pair of adjacent bits, we are able to calculate the transition probability of the bit-channels and locate the information bits and the frozen bits.

The organization of this section is as follows: In Section III-A, we first recall how to track the evolution of bit-channels in standard polar codes using the basic 2×2 transform. In Section III-B, we introduce a new transform and use it to establish a recursive relation between pairs of adjacent bits for standard polar codes. The purpose of Section III-B is to illustrate the application of the new transform in a familiar setting. In Section III-C, we use the new transform to track the evolution of adjacent bits in the ABS polar codes. The result in Section III-C accomplishes the second step of the ABS polar code construction, i.e., it allows us to locate the information bits and the frozen bits when the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in the recursive relation (7) are known. Next, in Section III-D, we explain how to pick these permutation matrices in the ABS polar code construction. Recall that the quantization operation is needed in the standard polar code construction [24] because the output alphabet size of the bit-channels grows exponentially with n . The same issue also arises in the ABS polar code construction, and we will discuss this in Section III-E. Finally, we put everything together and summarize the code construction algorithm for ABS polar codes in Section III-F.

A. Tracking the evolution of bit-channels in standard polar codes using the 2×2 transform

Let us first recall the 2×2 transform in the standard polar code construction.

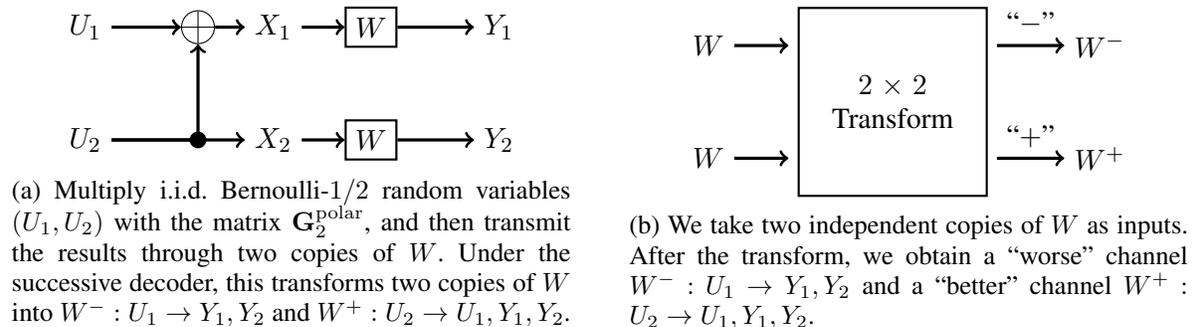


Fig. 1: The 2×2 basic polar transform

Given a BMS channel $W : \{0, 1\} \rightarrow \mathcal{Y}$, the transition probabilities of $W^- : \{0, 1\} \rightarrow \mathcal{Y}^2$ and $W^+ : \{0, 1\} \rightarrow \{0, 1\} \times \mathcal{Y}^2$ in Fig. 1 are given by

$$W^-(y_1, y_2|u_1) = \frac{1}{2} \sum_{u_2 \in \{0, 1\}} W(y_1|u_1 + u_2)W(y_2|u_2) \quad \text{for } u_1 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}, \quad (12)$$

$$W^+(u_1, y_1, y_2|u_2) = \frac{1}{2} W(y_1|u_1 + u_2)W(y_2|u_2) \quad \text{for } u_1, u_2 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}.$$

The basic 2×2 transform plays a fundamental role in the standard polar code construction because it allows us to efficiently track the evolution of bit-channels in a recursive way. More specifically, the bit-channels induced by the matrix $\mathbf{G}_n^{\text{polar}}$ are defined in Fig. 2 below. It is well known that the bit-channels associated with $\mathbf{G}_n^{\text{polar}}$ and the bit-channels associated with $\mathbf{G}_{n/2}^{\text{polar}}$ satisfy the following recursive relation:

$$W_{2i-1}^{(n)} = (W_i^{(n/2)})^- \quad \text{and} \quad W_{2i}^{(n)} = (W_i^{(n/2)})^+ \quad \text{for } 1 \leq i \leq n/2. \quad (13)$$

Both the code construction and the decoding algorithm of standard polar codes rely on this recursive relation.

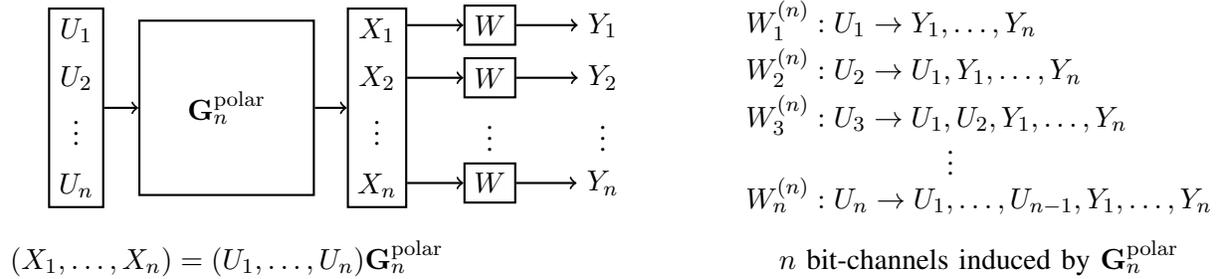


Fig. 2: U_1, \dots, U_n are $n = 2^m$ i.i.d. Bernoulli-1/2 random variables. $(X_1, \dots, X_n) = (U_1, \dots, U_n)\mathbf{G}_n^{\text{polar}}$ is the codeword vector, and (Y_1, \dots, Y_n) is the channel output vector. The n bit-channels induced by $\mathbf{G}_n^{\text{polar}}$ are listed on the right side of the figure. $W_i^{(n)}$ is the bit-channel mapping from U_i to $U_1, \dots, U_{i-1}, Y_1, \dots, Y_n$.

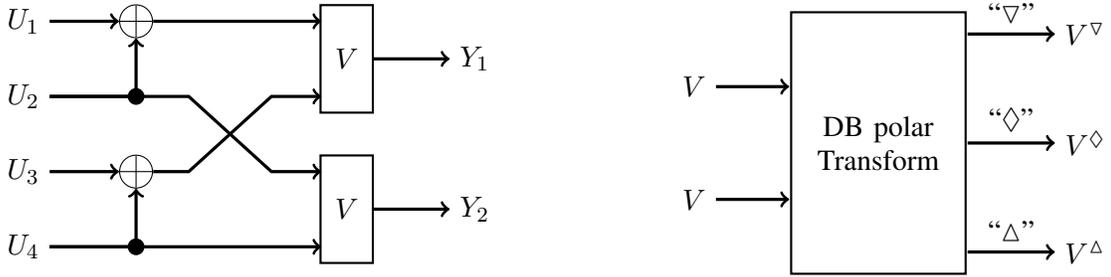
B. Tracking the evolution of adjacent bits in standard polar codes using a new transform

In the construction of ABS polar codes, we need to track the joint distribution of every pair of adjacent bits, not just the distribution of every single bit given the previous bits and channel outputs. To that end, we introduce a new transform, named as the Double-Bits (DB) polar transform. All the channels involved in the DB polar transform have 4-ary inputs. To distinguish between binary-input channels and 4-ary-input channels, we use W to denote the former channels and use V to denote latter channels². The details of the DB polar transform are illustrated in Fig. 3. Given a 4-ary-input channel $V : \{0, 1\}^2 \rightarrow \mathcal{Y}$, the transition

²More precisely, W and its variations such as $W^+, W^-, W_i^{(n)}$ are used for binary-input channels; V and its variations such as $V^\nabla, V^\diamond, V^\Delta, V_i^{(n)}$ are used for channels with 4-ary inputs.

probabilities of $V^\nabla : \{0, 1\}^2 \rightarrow \mathcal{Y}^2$, $V^\diamond : \{0, 1\}^2 \rightarrow \{0, 1\} \times \mathcal{Y}^2$, and $V^\Delta : \{0, 1\}^2 \rightarrow \{0, 1\}^2 \times \mathcal{Y}^2$ in Fig. 3 are given by

$$\begin{aligned}
 V^\nabla(y_1, y_2|u_1, u_2) &= \frac{1}{4} \sum_{u_3, u_4 \in \{0, 1\}} V(y_1|u_1 + u_2, u_3 + u_4)V(y_2|u_2, u_4) \\
 &\quad \text{for } u_1, u_2 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}, \\
 V^\diamond(u_1, y_1, y_2|u_2, u_3) &= \frac{1}{4} \sum_{u_4 \in \{0, 1\}} V(y_1|u_1 + u_2, u_3 + u_4)V(y_2|u_2, u_4) \\
 &\quad \text{for } u_1, u_2, u_3 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}, \\
 V^\Delta(u_1, u_2, y_1, y_2|u_3, u_4) &= \frac{1}{4} V(y_1|u_1 + u_2, u_3 + u_4)V(y_2|u_2, u_4) \\
 &\quad \text{for } u_1, u_2, u_3, u_4 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}.
 \end{aligned} \tag{14}$$



(a) U_1, U_2, U_3, U_4 are i.i.d. Bernoulli-1/2 random variables. The channel $V : \{0, 1\}^2 \rightarrow \mathcal{Y}$ takes two bits as its inputs, i.e., V has 4-ary inputs. Under the successive decoder, we have the following three channels: (1) $V^\nabla : U_1, U_2 \rightarrow Y_1, Y_2$; (2) $V^\diamond : U_2, U_3 \rightarrow U_1, Y_1, Y_2$; (3) $V^\Delta : U_3, U_4 \rightarrow U_1, U_2, Y_1, Y_2$.

(b) Two independent copies of V are transformed into three channels $V^\nabla, V^\diamond, V^\Delta$. These three channels also have 4-ary inputs. Note that the inputs of V^∇ and V^\diamond have one-bit overlap, and the inputs of V^\diamond and V^Δ also have one-bit overlap.

Fig. 3: The Double-Bits (DB) polar transform

The role of the DB polar transform in the construction of ABS polar codes is the same as the role of the 2×2 basic polar transform in the standard polar code construction. Instead of jumping directly into the ABS polar code construction, let us first use standard polar codes to illustrate how to track the evolution of adjacent bits recursively using the DB polar transform. In order to calculate the joint distribution of adjacent bits, we introduce the notion of adjacent-bits-channels, which is the counterpart of the bit-channels used for tracking the distribution of every single bit. We still use the setting in Fig. 2, where we defined the bit-channels. For the matrix $\mathbf{G}_n^{\text{polar}}$ and a BMS channel W , we define $n - 1$ adjacent-bits-channels $V_1^{(n)}, V_2^{(n)}, \dots, V_{n-1}^{(n)}$ as follows:

$$V_i^{(n)} : U_i, U_{i+1} \rightarrow U_1, \dots, U_{i-1}, Y_1, \dots, Y_n \quad \text{for } 1 \leq i \leq n - 1, \tag{15}$$

where $U_1, \dots, U_n, Y_1, \dots, Y_n$ are defined in Fig. 2. By definition, $V_1^{(n)}, V_2^{(n)}, \dots, V_{n-1}^{(n)}$ take two bits as their inputs, i.e., all of them have 4-ary inputs. Moreover, these adjacent-bits-channels depend on the BMS channel W , although we omit this dependence in the notation.

The following lemma allows us to calculate $V_1^{(n)}, V_2^{(n)}, \dots, V_{n-1}^{(n)}$ recursively from $V_1^{(n/2)}, V_2^{(n/2)}, \dots, V_{n/2-1}^{(n/2)}$.

Lemma 1. *Let $n \geq 4$. We have*

$$V_{2i-1}^{(n)} = (V_i^{(n/2)})^\nabla, \quad V_{2i}^{(n)} = (V_i^{(n/2)})^\diamond, \quad V_{2i+1}^{(n)} = (V_i^{(n/2)})^\Delta \quad \text{for } 1 \leq i \leq n/2 - 1. \tag{16}$$

The proof of Lemma 1 is given in Appendix A. The relation (16) is similar in nature to the relation (13), and the proof of (16) also uses the same method as the proof of (13). There is, however, one difference between these two recursive relations: The “+” and “−” transforms of different bit-channels are distinct while the “∇”, “◊” and “Δ” transforms of different adjacent-bits-channels may overlap. More precisely, the $n/2$ sets $\{(W_i^{(n/2)})_-, (W_i^{(n/2)})_+\}_{i=1}^{n/2}$ are disjoint while the two sets $\{(V_i^{(n/2)})_\nabla, (V_i^{(n/2)})_\diamond, (V_i^{(n/2)})_\Delta\}$ and $\{(V_{i+1}^{(n/2)})_\nabla, (V_{i+1}^{(n/2)})_\diamond, (V_{i+1}^{(n/2)})_\Delta\}$ have the following element in their intersection for every $1 \leq i \leq n/2 - 2$:

$$V_{2i+1}^{(n)} = (V_i^{(n/2)})_\Delta = (V_{i+1}^{(n/2)})_\nabla. \quad (17)$$

This gives us two methods of calculating $V_{2i+1}^{(n)}$ recursively for $1 \leq i \leq n/2 - 2$.

Lemma 1 tells us how to calculate $\{V_i^{(n)}\}_{i=1}^{n-1}$ from $\{V_i^{(n/2)}\}_{i=1}^{n/2-1}$ recursively for $n \geq 4$. The last question we need to answer is how to calculate the adjacent-bits-channel $V_1^{(2)}$ from the BMS channel W , because $V_1^{(2)}$ is the starting point of the recursive relation in Lemma 1. Fortunately, this is an easy task. Let us go back to the setting in Fig. 1. Given a BMS channel W , the adjacent-bits-channel $V_1^{(2)}$ is simply the channel mapping from U_1, U_2 to Y_1, Y_2 . More precisely, we have

$$V_1^{(2)}(y_1, y_2 | u_1, u_2) = W(y_1 | u_1 + u_2)W(y_2 | u_2). \quad (18)$$

After obtaining the transition probabilities of the adjacent-bits-channels $\{V_i^{(n)}\}_{i=1}^{n-1}$, it is straightforward to calculate the transition probabilities of the bit-channels $\{W_i^{(n)}\}_{i=1}^n$. More precisely, we have

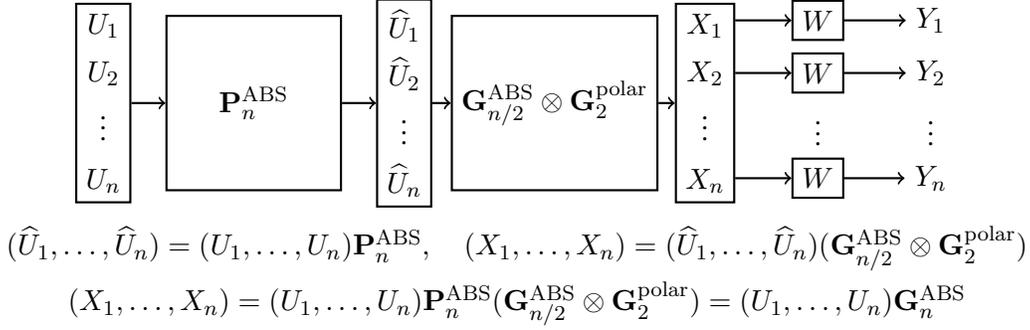
$$\begin{aligned} W_i^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{i-1} | u_i) &= \frac{1}{2} \sum_{u_{i+1} \in \{0,1\}} V_i^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{i-1} | u_i, u_{i+1}), \\ W_{i+1}^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_i | u_{i+1}) &= \frac{1}{2} V_i^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{i-1} | u_i, u_{i+1}) \end{aligned} \quad (19)$$

for $1 \leq i \leq n - 1$.

As a final remark, we note that the output alphabet size of the adjacent-bits-channels $\{V_i^{(n)}\}_{i=1}^{n-1}$ grows exponentially with n . Therefore, accurate calculations of $\{V_i^{(n)}\}_{i=1}^{n-1}$ are intractable. We need to quantize the output alphabets by merging output symbols with similar posterior distributions. Recall that in the standard polar code construction [24], we also need the quantization operation to calculate an approximation of the bit-channels $\{W_i^{(n)}\}_{i=1}^n$. Our quantization method is different from the one used in [24] because the adjacent-bits-channels have 4-ary inputs while the bit-channels have binary inputs. We will present our quantization method later in Section III-E.

C. Tracking the evolution of adjacent bits in ABS polar codes

As discussed at the beginning of this section, the construction of ABS polar codes consists of two main steps. The first step is to pick the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in the recursive relation (7), and the second step is to find which bits are information bits and which bits are frozen bits after picking these permutation matrices. In this subsection, we explain how to accomplish the second step. More precisely, we define the bit-channels and the adjacent-bits-channels for ABS polar codes in Fig. 4. The task of this subsection is to show how to calculate the capacity of the bit-channels $\{W_i^{(n), \text{ABS}}\}_{i=1}^n$ when the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in (7) are known. Then the information bits are simply the U_i 's satisfying that $I(W_i^{(n), \text{ABS}}) \approx 1$, where $I(\cdot)$ is the channel capacity. Unlike the standard polar codes, there does not exist a recursive relation between the bit-channels $\{W_i^{(n), \text{ABS}}\}_{i=1}^n$ and $\{W_i^{(n/2), \text{ABS}}\}_{i=1}^{n/2}$ for ABS polar codes. Instead, we derive a recursive relation between the adjacent-bits-channels $\{V_i^{(n), \text{ABS}}\}_{i=1}^{n-1}$ and $\{V_i^{(n/2), \text{ABS}}\}_{i=1}^{n/2-1}$. After that, the transition probabilities of $\{W_i^{(n), \text{ABS}}\}_{i=1}^n$ can be calculated from the transition probabilities of $\{V_i^{(n), \text{ABS}}\}_{i=1}^{n-1}$.



Two sets of bit-channels

$$\begin{aligned} \{W_i^{(n),\text{ABS}} : U_i \rightarrow U_1, \dots, U_{i-1}, Y_1, \dots, Y_n\}_{i=1}^n \\ \{\widehat{W}_i^{(n),\text{ABS}} : \widehat{U}_i \rightarrow \widehat{U}_1, \dots, \widehat{U}_{i-1}, Y_1, \dots, Y_n\}_{i=1}^n \end{aligned}$$

Two sets of adjacent-bits-channels

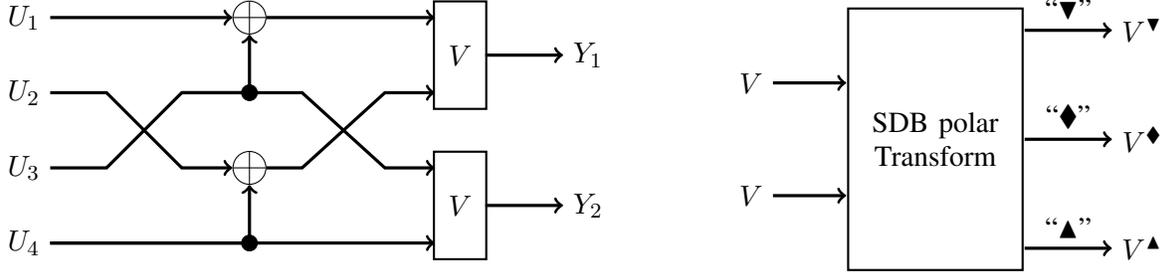
$$\begin{aligned} \{V_i^{(n),\text{ABS}} : U_i, U_{i+1} \rightarrow U_1, \dots, U_{i-1}, Y_1, \dots, Y_n\}_{i=1}^{n-1} \\ \{\widehat{V}_i^{(n),\text{ABS}} : \widehat{U}_i, \widehat{U}_{i+1} \rightarrow \widehat{U}_1, \dots, \widehat{U}_{i-1}, Y_1, \dots, Y_n\}_{i=1}^{n-1} \end{aligned}$$

Fig. 4: U_1, \dots, U_n are $n = 2^m$ i.i.d. Bernoulli-1/2 random variables. $(X_1, \dots, X_n) = (U_1, \dots, U_n)\mathbf{G}_n^{\text{ABS}}$ is the codeword vector, and (Y_1, \dots, Y_n) is the channel output vector. We view each Kronecker product with $\mathbf{G}_2^{\text{polar}}$ as one layer of polar transform and view each multiplication with a permutation matrix as one layer of permutation. Then $\mathbf{G}_n^{\text{ABS}}$ is obtained from m layers of polar transforms and m layers of permutations while $\mathbf{G}_{n/2}^{\text{ABS}} \otimes \mathbf{G}_2^{\text{polar}}$ is obtained from m layers of polar transforms and $m - 1$ layers of permutations. Therefore, $\{W_i^{(n),\text{ABS}}\}_{i=1}^n$ and $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ are the bit-channels and adjacent-bits-channels seen by the successive decoder after m layers of polar transforms and m layers of permutations. Similarly, $\{\widehat{W}_i^{(n),\text{ABS}}\}_{i=1}^n$ and $\{\widehat{V}_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ are the bit-channels and adjacent-bits-channels seen by the successive decoder after m layers of polar transforms and $m - 1$ layers of permutations.

In order to derive the recursive relation between the adjacent-bits-channels for ABS polar codes, we need another new transform named as the Swapped-Double-Bits (SDB) polar transform in addition to the DB polar transform defined in (14). The details of the SDB polar transform are illustrated in Fig. 5. In fact, the SDB polar transform is very similar to the DB polar transform. By comparing Fig. 3a and Fig. 5a, we can see that the only difference between these two transforms is the order of U_2 and U_3 . Given a 4-ary-input channel $V : \{0, 1\}^2 \rightarrow \mathcal{Y}$, the transition probabilities of $V^\blacktriangledown : \{0, 1\}^2 \rightarrow \mathcal{Y}^2$, $V^\blacklozenge : \{0, 1\}^2 \rightarrow \{0, 1\} \times \mathcal{Y}^2$, and $V^\blacktriangle : \{0, 1\}^2 \rightarrow \{0, 1\}^2 \times \mathcal{Y}^2$ in Fig. 5 are given by

$$\begin{aligned} V^\blacktriangledown(y_1, y_2 | u_1, u_2) &= \frac{1}{4} \sum_{u_3, u_4 \in \{0, 1\}} V(y_1 | u_1 + u_3, u_2 + u_4) V(y_2 | u_3, u_4) \\ &\quad \text{for } u_1, u_2 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}, \\ V^\blacklozenge(u_1, y_1, y_2 | u_2, u_3) &= \frac{1}{4} \sum_{u_4 \in \{0, 1\}} V(y_1 | u_1 + u_3, u_2 + u_4) V(y_2 | u_3, u_4) \\ &\quad \text{for } u_1, u_2, u_3 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}, \\ V^\blacktriangle(u_1, u_2, y_1, y_2 | u_3, u_4) &= \frac{1}{4} V(y_1 | u_1 + u_3, u_2 + u_4) V(y_2 | u_3, u_4) \\ &\quad \text{for } u_1, u_2, u_3, u_4 \in \{0, 1\} \text{ and } y_1, y_2 \in \mathcal{Y}. \end{aligned} \tag{20}$$

Recall that we use the set $\mathcal{I}^{(n)} = \{i_1, i_2, \dots, i_s\}$ to represent the permutation matrix $\mathbf{P}_n^{\text{ABS}}$ in (10). Moreover, we require that i_1, i_2, \dots, i_s in the set $\mathcal{I}^{(n)}$ satisfy the condition (11) because oth-



(a) U_1, U_2, U_3, U_4 are i.i.d. Bernoulli-1/2 random variables. The channel $V : \{0, 1\}^2 \rightarrow \mathcal{Y}$ takes two bits as its inputs, i.e., V has 4-ary inputs. Under the successive decoder, we have the following three channels: (1) $V^\nabla : U_1, U_2 \rightarrow Y_1, Y_2$; (2) $V^\blacklozenge : U_2, U_3 \rightarrow U_1, Y_1, Y_2$; (3) $V^\blacktriangle : U_3, U_4 \rightarrow U_1, U_2, Y_1, Y_2$.

(b) Two independent copies of V are transformed into three channels $V^\nabla, V^\blacklozenge, V^\blacktriangle$. These three channels also have 4-ary inputs. Note that the inputs of V^∇ and V^\blacklozenge have one-bit overlap, and the inputs of V^\blacklozenge and V^\blacktriangle also have one-bit overlap.

Fig. 5: The Swapped-Double-Bits (SDB) polar transform

erwise there does not exist a recursive relation between the adjacent-bits-channels $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ and $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$. We will give a detailed explanation about this later in Section III-G. Here we point out another property of the elements i_1, i_2, \dots, i_s in $\mathcal{I}^{(n)}$: they must all be even numbers. To see this, let us go back to the setting in Fig. 4. The role of $\mathcal{I}^{(n)}$ is to decide which pairs of adjacent bits to swap in the vector $(\widehat{U}_1, \widehat{U}_2, \dots, \widehat{U}_n)$ defined in Fig. 4. According to the discussion in Section II-B, we swap the adjacent bits \widehat{U}_i and \widehat{U}_{i+1} only if they are unordered, i.e., if \widehat{U}_i is more reliable than \widehat{U}_{i+1} under the successive decoder. In other words, we swap the adjacent bits \widehat{U}_i and \widehat{U}_{i+1} only if $I(\widehat{W}_i^{(n),\text{ABS}}) \geq I(\widehat{W}_{i+1}^{(n),\text{ABS}})$, where the bit-channels $\widehat{W}_i^{(n),\text{ABS}}$ and $\widehat{W}_{i+1}^{(n),\text{ABS}}$ are also defined in Fig. 4. Since $\{\widehat{W}_i^{(n),\text{ABS}}\}_{i=1}^n$ are obtained from the 2×2 basic polar transform of $\{W_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2}$, they satisfy the following relation:

$$\widehat{W}_{2i-1}^{(n),\text{ABS}} = (W_i^{(n/2),\text{ABS}})^- \quad \text{and} \quad \widehat{W}_{2i}^{(n),\text{ABS}} = (W_i^{(n/2),\text{ABS}})^+ \quad \text{for } 1 \leq i \leq n/2.$$

Therefore,

$$I(\widehat{W}_{2i-1}^{(n),\text{ABS}}) \leq I(W_i^{(n/2),\text{ABS}}) \leq I(\widehat{W}_{2i}^{(n),\text{ABS}}),$$

so we should not swap \widehat{U}_{2i-1} and \widehat{U}_{2i} for any $1 \leq i \leq n/2$. Thus we conclude that the set $\mathcal{I}^{(n)}$ in (10) only contains even numbers. Therefore, the elements of $\mathcal{I}^{(n)}$ can be written as $\mathcal{I}^{(n)} = \{2j_1, 2j_2, \dots, 2j_s\}$, and the condition (11) becomes

$$j_2 \geq j_1 + 2, \quad j_3 \geq j_2 + 2, \quad j_4 \geq j_3 + 2, \quad \dots, \quad j_s \geq j_{s-1} + 2. \quad (21)$$

Now we are ready to state the recursive relation between $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ and $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$.

Lemma 2. *Let $n \geq 4$. We write $\mathbf{P}_n^{\text{ABS}}$ in the form of (10) and require that $\mathcal{I}^{(n)} = \{2j_1, 2j_2, \dots, 2j_s\}$ satisfies (21). For $1 \leq i \leq n/2 - 1$, we have the following results:*

Case i) *If $2i \in \mathcal{I}^{(n)}$, then*

$$V_{2i-1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\nabla, \quad V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\blacklozenge, \quad V_{2i+1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\blacktriangle.$$

Case ii) *If $2(i-1) \in \mathcal{I}^{(n)}$ and $2(i+1) \in \mathcal{I}^{(n)}$, then*

$$V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\diamond.$$

Case iii) *If $2(i-1) \in \mathcal{I}^{(n)}$ and $2(i+1) \notin \mathcal{I}^{(n)}$, then*

$$V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\diamond, \quad V_{2i+1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\Delta.$$

Case iv) If $2(i-1) \notin \mathcal{I}^{(n)}$ and $2(i+1) \in \mathcal{I}^{(n)}$, then

$$V_{2i-1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\nabla, \quad V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\diamond.$$

Case v) If $2(i-1) \notin \mathcal{I}^{(n)}$, $2i \notin \mathcal{I}^{(n)}$ and $2(i+1) \notin \mathcal{I}^{(n)}$, then

$$V_{2i-1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\nabla, \quad V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\diamond, \quad V_{2i+1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\Delta.$$

Note that in a previous arXiv version and the ISIT version [1] of this paper, the statement of this lemma was not complete. In the previous versions, **Case ii)** was missing, and the conditions in **Case iii)** and **Case iv)** were incomplete.

The proof of Lemma 2 is omitted because it is essentially the same as the proof of Lemma 1. Here we point out one difference between Lemma 1 and Lemma 2. Lemma 1 tells us that $V_{2i+1}^{(n)}$ can be recursively calculated in two different ways for every $1 \leq i \leq n/2 - 2$; see (17). However, for $i \in \{j_1 - 1, j_2 - 1, \dots, j_s - 1\} \cup \{j_1, j_2, \dots, j_s\}$, there is only one way to calculate $V_{2i+1}^{(n),\text{ABS}}$ recursively. More precisely, if $i \in \{j_1 - 1, j_2 - 1, \dots, j_s - 1\}$, then $V_{2i+1}^{(n),\text{ABS}}$ can only be calculated from $V_{2i+1}^{(n),\text{ABS}} = (V_{i+1}^{(n/2),\text{ABS}})^\blacktriangledown$, and the relation $V_{2i+1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\Delta$ does *not* hold. Similarly, if $i \in \{j_1, j_2, \dots, j_s\}$, then $V_{2i+1}^{(n),\text{ABS}}$ can only be calculated from $V_{2i+1}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\blacktriangle$, and the relation $V_{2i+1}^{(n),\text{ABS}} = (V_{i+1}^{(n/2),\text{ABS}})^\nabla$ does *not* hold.

Since we require $n \geq 4$ in Lemma 2, the starting point of the recursive relation in Lemma 2 is $V_1^{(2),\text{ABS}}$. It is easy to see that the permutation matrix $\mathbf{P}_2^{\text{ABS}}$ is the identity matrix. Therefore, given a BMS channel W , the transition probability of $V_1^{(2),\text{ABS}}$ is given by

$$V_1^{(2),\text{ABS}}(y_1, y_2 | u_1, u_2) = W(y_1 | u_1 + u_2)W(y_2 | u_2). \quad (22)$$

Note that this is the same as (18) for standard polar codes.

After obtaining the transition probabilities of the adjacent-bits-channels $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$, we can use (19) to calculate the transition probabilities of the bit-channels $\{W_i^{(n),\text{ABS}}\}_{i=1}^n$. We only need to replace $W_i^{(n)}$, $W_{i+1}^{(n)}$, $V_i^{(n)}$ in (19) with $W_i^{(n),\text{ABS}}$, $W_{i+1}^{(n),\text{ABS}}$, $V_i^{(n),\text{ABS}}$. Once the transition probabilities of $\{W_i^{(n),\text{ABS}}\}_{i=1}^n$ are known, we are able to determine which bits are information bits and which bits are frozen bits.

D. Constructing the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in (7)

We construct the permutation matrices in (7) one by one, starting from $\mathbf{P}_2^{\text{ABS}}$. Therefore, the matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_{n/2}^{\text{ABS}}$ are already known when we construct $\mathbf{P}_n^{\text{ABS}}$. The method described in Section III-C allows us to calculate the transition probabilities of the adjacent-bits-channels $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$ from $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_{n/2}^{\text{ABS}}$. As a consequence, we know the transition probabilities of $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$ when constructing $\mathbf{P}_n^{\text{ABS}}$. Since the set $\mathcal{I}^{(n)} = \{2j_1, 2j_2, \dots, 2j_s\}$ in (10) uniquely determines $\mathbf{P}_n^{\text{ABS}}$, constructing $\mathbf{P}_n^{\text{ABS}}$ is further equivalent to constructing the set $\mathcal{S}^* = \{j_1, j_2, \dots, j_s\}$, where the elements j_1, j_2, \dots, j_s satisfy the condition (21).

Before presenting how to construct the set \mathcal{S}^* , let us introduce some notation. Suppose that $V : \{0, 1\}^2 \rightarrow \mathcal{Y}$ is an adjacent-bits-channel with 4-ary inputs. Define two bit-channels $V_{\text{first}} : \{0, 1\} \rightarrow \mathcal{Y}$ and $V_{\text{second}} : \{0, 1\} \rightarrow \{0, 1\} \times \mathcal{Y}$ as

$$V_{\text{first}}(y|u_1) = \frac{1}{2} \sum_{u_2 \in \{0,1\}} V(y|u_1, u_2) \quad \text{and} \quad V_{\text{second}}(y, u_1|u_2) = \frac{1}{2} V(y|u_1, u_2).$$

Comparing this with (19), we can see that if V is $V_i^{(n)}$, then V_{first} is simply $W_i^{(n)}$, and V_{second} is $W_{i+1}^{(n)}$. Similarly, if V is $V_i^{(n),\text{ABS}}$, then V_{first} is simply $W_i^{(n),\text{ABS}}$, and V_{second} is $W_{i+1}^{(n),\text{ABS}}$. Next we define

$$I_{\text{first}}(V) := I(V_{\text{first}}) \quad \text{and} \quad I_{\text{second}}(V) := I(V_{\text{second}}),$$

$$g(V) := I_{\text{first}}(V)(1 - I_{\text{first}}(V)) + I_{\text{second}}(V)(1 - I_{\text{second}}(V)).$$

The function $g(V)$ measures the polarization level of the two bit-channels induced by V . In particular, $g(V) \approx 0$ means that the capacity of both bit-channels is very close to either 0 or 1. Finally, for $1 \leq i \leq n/2 - 1$, we define

$$\text{Score}(i) := g((V_i^{(n/2),\text{ABS}})^\diamond) - g((V_i^{(n/2),\text{ABS}})^\blacklozenge).$$

The interpretation of $\text{Score}(i)$ is as follows: According to Lemma 2, if $i \in \mathcal{S}^*$, then $V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\blacklozenge$; if $i \notin \mathcal{S}^*$, then $V_{2i}^{(n),\text{ABS}} = (V_i^{(n/2),\text{ABS}})^\diamond$. Therefore, $g((V_i^{(n/2),\text{ABS}})^\blacklozenge)$ measures the polarization level of the two bit-channels $W_{2i}^{(n),\text{ABS}}$ and $W_{2i+1}^{(n),\text{ABS}}$ when we include i in the set \mathcal{S}^* . Similarly, $g((V_i^{(n/2),\text{ABS}})^\diamond)$ measures the polarization level of the two bit-channels $W_{2i}^{(n),\text{ABS}}$ and $W_{2i+1}^{(n),\text{ABS}}$ when we do not include i in the set \mathcal{S}^* . If $\text{Score}(i) > 0$, then including i in the set \mathcal{S}^* accelerates polarization. If $\text{Score}(i) < 0$, then including i in the set \mathcal{S}^* slows down polarization, and in this case we should not include i in \mathcal{S}^* .

If we ignore the condition (21), then we can simply choose the set \mathcal{S}^* to be $\mathcal{S}^* = \{i : \text{Score}(i) > 0\}$. However, as we will see in Section III-G, the condition (21) is crucial for us to calculate the transition probabilities of the adjacent-bits-channels, so it must be satisfied. As a consequence, we need to find a set $\mathcal{S}^* \subseteq \{1, 2, \dots, n/2 - 1\}$ to maximize $\sum_{i \in \mathcal{S}^*} \text{Score}(i)$ under the constraint that the distance between any two distinct elements of \mathcal{S}^* must be at least 2. In other words, we need to solve the following optimization problem:

$$\begin{aligned} \mathcal{S}^* &= \operatorname{argmax}_{\mathcal{S} \subseteq \{1, 2, \dots, n/2-1\}} \sum_{i \in \mathcal{S}} \text{Score}(i) \\ &\text{subject to: } |i_1 - i_2| \geq 2 \text{ for all } i_1, i_2 \in \mathcal{S} \text{ such that } i_1 \neq i_2. \end{aligned} \quad (23)$$

This problem can be solved using a dynamic programming method. For $1 \leq j \leq n/2 - 1$, define

$$\begin{aligned} \mathcal{S}_j^* &= \operatorname{argmax}_{\mathcal{S} \subseteq \{1, 2, \dots, j\}} \sum_{i \in \mathcal{S}} \text{Score}(i) \\ &\text{subject to: } |i_1 - i_2| \geq 2 \text{ for all } i_1, i_2 \in \mathcal{S} \text{ such that } i_1 \neq i_2, \\ M_j &= \max_{\mathcal{S} \subseteq \{1, 2, \dots, j\}} \sum_{i \in \mathcal{S}} \text{Score}(i) \\ &\text{subject to: } |i_1 - i_2| \geq 2 \text{ for all } i_1, i_2 \in \mathcal{S} \text{ such that } i_1 \neq i_2. \end{aligned}$$

By definition, we can see that $M_1 \leq M_2 \leq M_3 \leq \dots \leq M_{n/2-1}$. The sets $\mathcal{S}_1^*, \mathcal{S}_2^*$ and the maximum values M_1, M_2 can be calculated as follows: If $\text{Score}(1) > 0$, then $\mathcal{S}_1^* = \{1\}$ and $M_1 = \text{Score}(1)$. If $\text{Score}(1) \leq 0$, then $\mathcal{S}_1^* = \emptyset$ and $M_1 = 0$. If $\text{Score}(2) > M_1$, then $\mathcal{S}_2^* = \{2\}$ and $M_2 = \text{Score}(2)$. If $\text{Score}(2) \leq M_1$, then $\mathcal{S}_2^* = \mathcal{S}_1^*$ and $M_2 = M_1$. For $j \geq 3$, the set \mathcal{S}_j^* and the maximum value M_j can be calculated recursively as follows: If $\text{Score}(j) + M_{j-2} > M_{j-1}$, then $\mathcal{S}_j^* = \mathcal{S}_{j-2}^* \cup \{j\}$ and $M_j = \text{Score}(j) + M_{j-2}$. If $\text{Score}(j) + M_{j-2} \leq M_{j-1}$, then $\mathcal{S}_j^* = \mathcal{S}_{j-1}^*$ and $M_j = M_{j-1}$. This dynamic programming algorithm allows us to calculate \mathcal{S}_j^* for every $1 \leq j \leq n/2 - 1$. In particular, we are able to calculate $\mathcal{S}_{n/2-1}^* = \mathcal{S}^*$, which is the set we want to construct. Once we know the set $\mathcal{S}^* = \{j_1, j_2, \dots, j_s\}$, we can immediately write out the set $\mathcal{I}^{(n)} = \{2j_1, 2j_2, \dots, 2j_s\}$ and obtain the corresponding permutation matrix $\mathbf{P}_n^{\text{ABS}}$ according to (10).

As a final remark, we note that $\mathbf{P}_2^{\text{ABS}}$ is always the identity matrix. However, for $n \geq 4$, the permutation matrix $\mathbf{P}_n^{\text{ABS}}$ depends on the underlying BMS channel W .

E. Quantization of the output alphabet

Algorithm 1: QuantizeChannel(μ, V)

Input: an upper bound μ on the output alphabet size after quantization; an adjacent-bits-channel V with outputs y_1, y_2, \dots, y_M

Output: quantized channel \tilde{V} with outputs $\{\tilde{y}_{i_1, i_2, i_3} : 0 \leq i_1, i_2, i_3 \leq b\}$

```

1 if  $M \leq \mu$  then
2   Set  $\tilde{V}$  to be the same as  $V$ 
3 else
4    $b \leftarrow \lfloor \mu^{1/3} \rfloor - 1$ 
5   Set  $\tilde{V}(\tilde{y}_{i_1, i_2, i_3} | (u_1, u_2)) = 0$  for all  $0 \leq i_1, i_2, i_3 \leq b$  and all  $u_1, u_2 \in \{0, 1\}$ 
6                                      $\triangleright$  Initialize all the transition probabilities of  $\tilde{V}$  as 0
7   for  $j = 1, 2, \dots, M$  do
8      $sum \leftarrow V(y_j | (0, 0)) + V(y_j | (0, 1)) + V(y_j | (1, 0)) + V(y_j | (1, 1))$ 
9      $p_1 \leftarrow \frac{V(y_j | (0, 0))}{sum}$ ,  $p_2 \leftarrow \frac{V(y_j | (0, 1))}{sum}$ ,  $p_3 \leftarrow \frac{V(y_j | (1, 0))}{sum}$ 
10                                          $\triangleright$  Calculate the posterior probability of  $y_j$ 
11      $i_1 \leftarrow \lfloor bp_1 \rfloor$ ,  $i_2 \leftarrow \lfloor bp_2 \rfloor$ ,  $i_3 \leftarrow \lfloor bp_3 \rfloor$ 
12      $\tilde{V}(\tilde{y}_{i_1, i_2, i_3} | (u_1, u_2)) \leftarrow \tilde{V}(\tilde{y}_{i_1, i_2, i_3} | (u_1, u_2)) + V(y_j | (u_1, u_2))$  for all  $u_1, u_2 \in \{0, 1\}$ 
13                                          $\triangleright$  Merge  $y_j$  into  $\tilde{y}_{i_1, i_2, i_3}$ 
14 return  $\tilde{V}$ 

```

An important step in the construction of standard polar codes is to quantize the output alphabets of the bit-channels $\{W_i^{(n)}\}_{i=1}^n$ because the output alphabet size grows exponentially with the code length n . The most widely used quantization method for binary-input standard polar codes was given in [24], where the main idea is to merge output symbols with similar posterior distributions using a greedy algorithm. This greedy algorithm was later generalized to construct polar codes with non-binary input alphabets [25]–[27]. The time complexity of the greedy quantization algorithm is $O(\mu^2 \log \mu)$, where μ is the maximum size of the output alphabet after quantization. Since there are $2n - 1$ bit-channels we need to quantize in the code construction procedure, the overall time complexity of standard polar code construction is $O(n\mu^2 \log \mu)$.

In the ABS polar code construction, the output alphabet size of the adjacent-bits-channels $\{V_i^{(n)}\}_{i=1}^{n-1}$ also grows exponentially with n , and the quantization operations are also needed. Since the adjacent-bits-channels have 4-ary inputs, we can simply use the greedy quantization algorithms proposed in [25]–[27] for polar codes with non-binary inputs. However, in practical implementations, we found that these greedy algorithms for non-binary inputs usually involve implicit large constants in their time complexity. Therefore, we propose a new quantization algorithm to merge the output symbols of the adjacent-bits-channels $\{V_i^{(n)}\}_{i=1}^{n-1}$. The time complexity of our new quantization algorithm is $O(\mu^2)$. Since there are $\Theta(n)$ adjacent-bits-channels we need to quantize in the ABS polar code construction, its overall time complexity is $O(n\mu^2)$.

Our new quantization algorithm works as follows. Given an upper bound μ on the output alphabet size after quantization, we define $b = \lfloor \mu^{1/3} \rfloor - 1$. For an adjacent-bits-channel V , we write its 4 inputs as $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$, and we write its outputs as y_1, y_2, \dots, y_M , where M is the output alphabet size of V . We use \tilde{V} to denote the channel after output quantization. The 4 inputs of \tilde{V} are the same as the original channel V , and the outputs of \tilde{V} are written as $\{\tilde{y}_{i_1, i_2, i_3} : 0 \leq i_1, i_2, i_3 \leq b\}$. Clearly, the output alphabet size of \tilde{V} is no larger than μ . With the above notation in mind, we present our quantization algorithm in Algorithm 1. In our implementation, we pick $\mu = 250000$.

F. Summary of the ABS polar code construction

In Section III-C, we showed how to calculate the transition probabilities of the adjacent-bits-channels $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ when the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in (7) are known. In Section III-D, we showed how to construct the permutation matrix $\mathbf{P}_n^{\text{ABS}}$ when the transition probabilities of $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$ are available. In Section III-E, we proposed Algorithm 1 to quantize the output alphabets of the adjacent-bits-channels. Now we are in a position to put everything together and present the code construction algorithm for ABS polar codes in Algorithm 2.

Algorithm 2: ABSConstruct(n, k, W)

Input: code length $n = 2^m \geq 4$, code dimension k , and the BMS channel W

Output: the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$, and the index set \mathcal{A} of the information bits

- 1 Quantize the output alphabet of W using the method in [24] ▷ This step is needed when the output alphabet size of W is very large, e.g., when W has a continuous output alphabet.
 - 2 Set $\mathbf{P}_2^{\text{ABS}}$ to be the identity matrix
 - 3 Calculate the transition probability of $V_1^{(2),\text{ABS}}$ from W using (22)
 - 4 Quantize the output alphabet of $V_1^{(2),\text{ABS}}$ using Algorithm 1
 - 5 **for** $n_0 = 4, 8, 16, \dots, n$ **do**
 - 6 Construct $\mathbf{P}_{n_0}^{\text{ABS}}$ from $\{V_i^{(n_0/2),\text{ABS}}\}_{i=1}^{n_0/2-1}$ using the method in Section III-D
 - 7 Calculate the transition probabilities of $\{V_i^{(n_0),\text{ABS}}\}_{i=1}^{n_0-1}$ from $\mathbf{P}_{n_0}^{\text{ABS}}$ and $\{V_i^{(n_0/2),\text{ABS}}\}_{i=1}^{n_0/2-1}$ using Lemma 2
 - 8 Quantize the output alphabets of $\{V_i^{(n_0),\text{ABS}}\}_{i=1}^{n_0-1}$ using Algorithm 1
 - 9 Calculate the transition probabilities of $\{W_i^{(n),\text{ABS}}\}_{i=1}^n$ from the transition probabilities of $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$.
 - 10 Sort the capacity of the bit-channels $\{W_i^{(n),\text{ABS}}\}_{i=1}^n$ to obtain $I(W_{i_1}^{(n),\text{ABS}}) \geq I(W_{i_2}^{(n),\text{ABS}}) \geq \dots \geq I(W_{i_n}^{(n),\text{ABS}})$, where $\{i_1, i_2, \dots, i_n\}$ is a permutation of $\{1, 2, \dots, n\}$
 - 11 $\mathcal{A} \leftarrow \{i_1, i_2, \dots, i_k\}$
 - 12 **return** $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}, \mathcal{A}$
-

G. Necessity of the condition (21)

The condition (21) is necessary for us to derive a recursive relation between $\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ and $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$. In order to prove this claim, we introduce some notation. Instead of (U_1, U_2, \dots, U_n) , now we use $(U_1^{(n)}, U_2^{(n)}, \dots, U_n^{(n)})$ to denote the message vector. We add the superscript (n) in the notation to distinguish between random variables in different layers. Define

$$(\widehat{U}_1^{(n)}, \widehat{U}_2^{(n)}, \dots, \widehat{U}_n^{(n)}) = (U_1^{(n)}, U_2^{(n)}, \dots, U_n^{(n)}) \mathbf{P}_n^{\text{ABS}}.$$

We further define random vectors $(U_{1,1}^{(n/2)}, U_{2,1}^{(n/2)}, \dots, U_{n/2,1}^{(n/2)})$ and $(U_{1,2}^{(n/2)}, U_{2,2}^{(n/2)}, \dots, U_{n/2,2}^{(n/2)})$ as follows:

$$U_{i,1}^{(n/2)} = \widehat{U}_{2i-1}^{(n)} + \widehat{U}_{2i}^{(n)}, \quad U_{i,2}^{(n/2)} = \widehat{U}_{2i}^{(n)},$$

i.e., the vectors $(U_{1,1}^{(n/2)}, U_{2,1}^{(n/2)}, \dots, U_{n/2,1}^{(n/2)})$ and $(U_{1,2}^{(n/2)}, U_{2,2}^{(n/2)}, \dots, U_{n/2,2}^{(n/2)})$ are obtained from applying one layer of polar transform to $(\widehat{U}_1^{(n)}, \widehat{U}_2^{(n)}, \dots, \widehat{U}_n^{(n)})$. By definition, $V_i^{(n),\text{ABS}}$ gives us the conditional distribution of $(U_i^{(n)}, U_{i+1}^{(n)})$ given the channel outputs and the previous message bits; $V_i^{(n/2),\text{ABS}}$ gives us the conditional distribution of $(U_{i,1}^{(n/2)}, U_{i+1,1}^{(n/2)})$ and the conditional distribution of $(U_{i,2}^{(n/2)}, U_{i+1,2}^{(n/2)})$ given the channel outputs and the previous message bits. Therefore, deriving a recursive relation between

$\{V_i^{(n),\text{ABS}}\}_{i=1}^{n-1}$ and $\{V_i^{(n/2),\text{ABS}}\}_{i=1}^{n/2-1}$ is equivalent to the following task: Suppose that we know the joint distribution³ of $(U_{i,j}^{(n/2)}, U_{i+1,j}^{(n/2)})$ for all $1 \leq i \leq n/2 - 1$ and $j \in \{1, 2\}$. The task is to calculate the joint distribution of $(U_i^{(n)}, U_{i+1}^{(n)})$ for all $1 \leq i \leq n - 1$. We will show that it is not possible to accomplish this task without the condition (21).

Suppose that the condition (21) does not hold. Then there exists an integer i such that we swap the adjacent bits $\widehat{U}_{2i}^{(n)}$ and $\widehat{U}_{2i+1}^{(n)}$, and we also swap $\widehat{U}_{2i+2}^{(n)}$ and $\widehat{U}_{2i+3}^{(n)}$; see Fig. 6 for an illustration. According to our assumption, we know the joint distribution of $(U_{i,1}^{(n/2)}, U_{i+1,1}^{(n/2)})$ and the joint distribution of $(U_{i,2}^{(n/2)}, U_{i+1,2}^{(n/2)})$. Moreover, $(U_{i,1}^{(n/2)}, U_{i+1,1}^{(n/2)})$ and $(U_{i,2}^{(n/2)}, U_{i+1,2}^{(n/2)})$ are independent. Therefore, we know the joint distribution of $(U_{i,1}^{(n/2)}, U_{i,2}^{(n/2)}, U_{i+1,1}^{(n/2)}, U_{i+1,2}^{(n/2)})$. Since there is a one-to-one mapping between $(\widehat{U}_{2i-1}^{(n)}, \widehat{U}_{2i}^{(n)}, \widehat{U}_{2i+1}^{(n)}, \widehat{U}_{2i+2}^{(n)})$ and $(U_{i,1}^{(n/2)}, U_{i,2}^{(n/2)}, U_{i+1,1}^{(n/2)}, U_{i+1,2}^{(n/2)})$, we also know the distribution of $(\widehat{U}_{2i-1}^{(n)}, \widehat{U}_{2i}^{(n)}, \widehat{U}_{2i+1}^{(n)}, \widehat{U}_{2i+2}^{(n)})$. Since $(U_{2i-1}^{(n)}, U_{2i}^{(n)}, U_{2i+1}^{(n)})$ is a function of $(\widehat{U}_{2i-1}^{(n)}, \widehat{U}_{2i}^{(n)}, \widehat{U}_{2i+1}^{(n)})$, we are able to calculate the joint distribution of $(U_{2i-1}^{(n)}, U_{2i}^{(n)})$ and the joint distribution of $(U_{2i}^{(n)}, U_{2i+1}^{(n)})$. Using a similar argument, we can show that we are able to calculate the joint distribution of $(U_{2i+2}^{(n)}, U_{2i+3}^{(n)})$ and the joint distribution of $(U_{2i+3}^{(n)}, U_{2i+4}^{(n)})$. The only problem is that we are not able to calculate the joint distribution of $(U_{2i+1}^{(n)}, U_{2i+2}^{(n)})$. By definition,

$$U_{2i+1}^{(n)} = \widehat{U}_{2i}^{(n)} = U_{i,2}^{(n/2)}, \quad U_{2i+2}^{(n)} = \widehat{U}_{2i+3}^{(n)} = U_{i+2,1}^{(n/2)} + U_{i+2,2}^{(n/2)}.$$

Therefore, our task is to calculate the joint distribution of $(U_{i,2}^{(n/2)}, U_{i+2,1}^{(n/2)} + U_{i+2,2}^{(n/2)})$. Since the two random vectors $(U_{1,1}^{(n/2)}, U_{2,1}^{(n/2)}, \dots, U_{n/2,1}^{(n/2)})$ and $(U_{1,2}^{(n/2)}, U_{2,2}^{(n/2)}, \dots, U_{n/2,2}^{(n/2)})$ are independent, this further requires us to know the joint distribution of $(U_{i,2}^{(n/2)}, U_{i+2,2}^{(n/2)})$, which is not available. Therefore, we are not able to calculate the joint distribution of $(U_{2i+1}^{(n)}, U_{2i+2}^{(n)})$. This proves the necessity of (21).

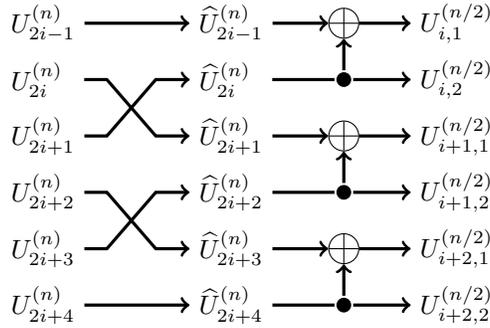


Fig. 6: Swap the adjacent bits \widehat{U}_{2i} and \widehat{U}_{2i+1} . Also swap \widehat{U}_{2i+2} and \widehat{U}_{2i+3} .

IV. THE ENCODING ALGORITHM FOR ABS POLAR CODES

In this section, we present the encoding algorithm of ABS polar codes. Suppose that we have constructed an (n, k) ABS polar code with permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \mathbf{P}_8^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ and the

³More precisely, this should be the conditional distribution of $(U_{i,j}^{(n/2)}, U_{i+1,j}^{(n/2)})$ given the channel outputs and the previous message bits. Similarly, the joint distribution of $(U_i^{(n)}, U_{i+1}^{(n)})$ in the next sentence also refers to the conditional distribution.

index set $\mathcal{A} = \{i_1, i_2, \dots, i_k\}$ of the information bits. We present the encoding algorithm of this code in Algorithm 3 below.

Algorithm 3: Encode((m_1, m_2, \dots, m_k))

Input: the message vector $(m_1, m_2, \dots, m_k) \in \{0, 1\}^k$
Output: the codeword $(c_1, c_2, \dots, c_n) \in \{0, 1\}^n$, where $n = 2^m$ is the code length

- 1 Initialize (c_1, c_2, \dots, c_n) as the all-zero vector
- 2 $(c_{i_1}, c_{i_2}, \dots, c_{i_k}) \leftarrow (m_1, m_2, \dots, m_k)$
- 3 ▷ Recall that i_1, i_2, \dots, i_k are the indices of the information bits.
- 4 **for** $i = 0, 1, 2, 3, \dots, m - 1$ **do**
- 5 $t \leftarrow 2^i$
- 6 $n_0 \leftarrow 2^{m-i}$
- 7 **for** $h = 1, 2, 3, \dots, t$ **do**
- 8 $(c_h, c_{h+t}, c_{h+2t}, c_{h+3t}, \dots, c_{h+(n_0-1)t}) \leftarrow (c_h, c_{h+t}, c_{h+2t}, c_{h+3t}, \dots, c_{h+(n_0-1)t}) \mathbf{P}_{n_0}^{\text{ABS}}$
- 9 ▷ Line 8 is the only difference between the encoding algorithms for ABS polar codes and standard polar codes
- 10 **for** $j = 0, 1, 2, 3, \dots, n_0/2 - 1$ **do**
- 11 $c_{h+2jt} \leftarrow c_{h+2jt} + c_{h+2jt+t}$
- 12 ▷ The addition between c_{h+2jt} and $c_{h+2jt+t}$ is over the binary field
- 13 **return** (c_1, c_2, \dots, c_n)

Without Line 8, Algorithm 3 is the same as the encoding algorithm of standard polar codes, whose time complexity is $O(n \log(n))$. In line 8, we perform a permutation on n_0 elements. According to our code construction, each of these n_0 elements is swapped at most once, so the number of operations involved in this permutation is no more than $n_0 = 2^{m-i}$. From the for loop in Line 7, we can see that Line 8 is executed $t = 2^i$ times for each $i \in \{0, 1, \dots, m - 1\}$. In other words, for each fixed value of i , Line 8 induces at most $n_0 * t = 2^m = n$ operations. Therefore, the total number of operations induced by Line 8 is upper bounded by $n * m = n \log(n)$. Thus we conclude that the encoding complexity of ABS polar codes is still $O(n \log(n))$.

Proposition 1. *The encoding time complexity of ABS polar codes is $O(n \log(n))$.*

Note that the set $\mathcal{I}^{(n)}$ in (10) uniquely determines the permutation matrix $\mathbf{P}_n^{\text{ABS}}$. In Fig. 7, we present the encoding circuit of an $(n = 16, k = 8)$ ABS polar code defined by the following sets:

$$\begin{aligned} \mathcal{I}^{(2)} &= \emptyset, & \mathcal{I}^{(4)} &= \emptyset, & \mathcal{I}^{(8)} &= \{4\}, & \mathcal{I}^{(16)} &= \{6, 10\}, \\ \mathcal{A} &= \{9, 10, 11, 12, 13, 14, 15, 16\}. \end{aligned} \tag{24}$$

V. THE SCL DECODER FOR ABS POLAR CODES

In this section, we present a new SCL decoder for ABS polar codes. The organization of this section is as follows: In Section V-A, we recap the classic SCL decoder for standard polar codes based on the 2×2 polar transform. The purpose of doing so is to get ourselves familiar with the recursive structure, which is shared by both the classic SCL decoder and our new SCL decoder. The SCL decoder presented in Section V-A is based on the one proposed in [28]. While the classic SCL decoder is based on the 2×2 polar transform, our new SCL decoder is based on the DB polar transform and the SDB polar transform; see Fig. 3 and Fig. 5 for the definitions of these two transforms. Instead of jumping directly into the decoding of ABS polar codes, we first present a new SCL decoder for standard polar codes based on the DB polar transform in Section V-B. This new SCL decoder for standard polar codes already contains most of the new ingredients in the SCL decoder for ABS polar codes, and it helps us learn these new ingredients in a familiar setting. Finally, in Section V-C, we present our new SCL decoder for ABS polar codes.

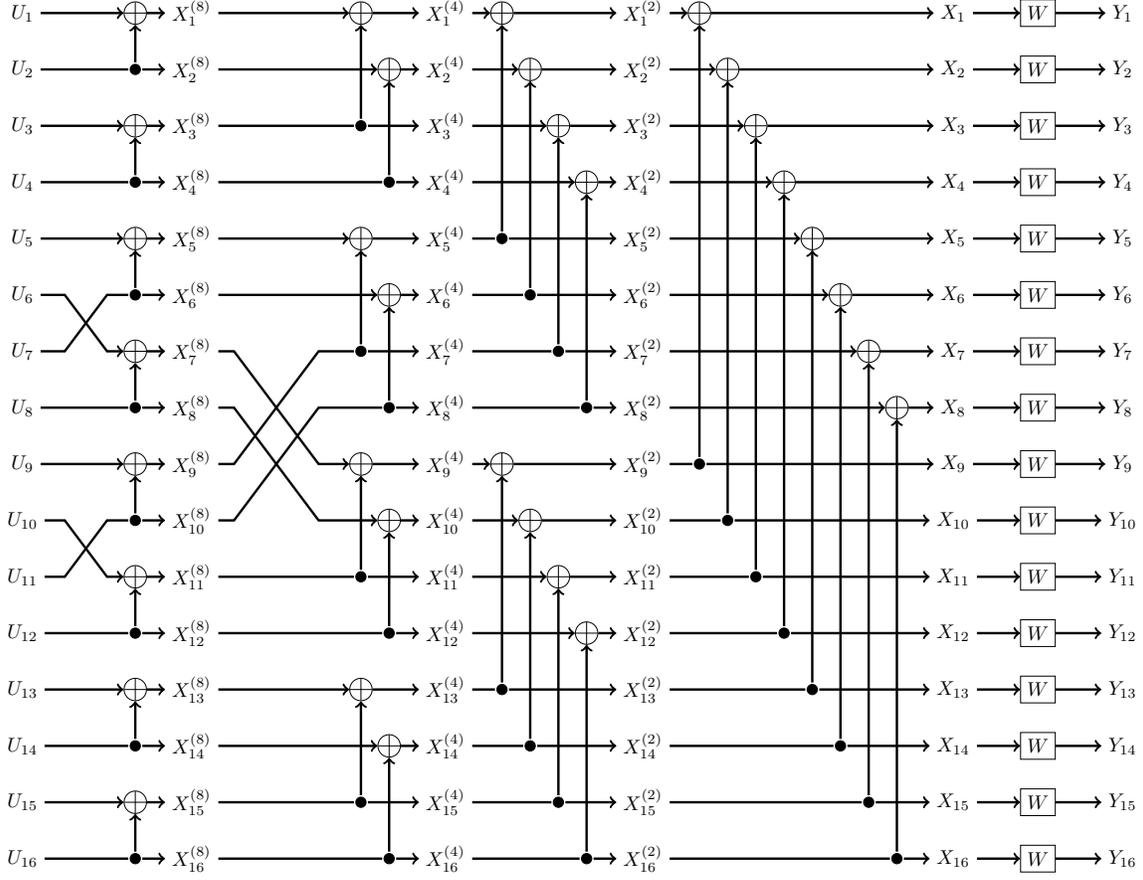


Fig. 7: Encoding circuit of an $(n = 16, k = 8)$ ABS polar code defined by the sets in (24).

A. SCL decoder for standard polar codes based on the 2×2 polar transform

In this subsection, we recap the classic SCL decoder proposed in [28] for standard polar codes. Suppose that the code length is $n = 2^m$, and the upper bound of the list size in the SCL decoder is L . We use $L_c \in \{1, 2, \dots, L\}$ to denote the current list size. \mathcal{A} is the index set of the information bits.

Before describing the decoding algorithms, let us introduce some notation and intermediate variables. Following the notation in Fig. 2, (U_1, U_2, \dots, U_n) is the message vector, and we use (X_1, \dots, X_n) and (Y_1, \dots, Y_n) to denote the random codeword vector and the random channel output vector, respectively. We use (y_1, \dots, y_n) to denote a realization of the random vector (Y_1, \dots, Y_n) . For each $0 \leq \lambda \leq m$, we introduce an intermediate vector $(X_1^{(2^\lambda)}, X_2^{(2^\lambda)}, \dots, X_n^{(2^\lambda)})$. For $\lambda = m$, we define the intermediate vector as

$$(X_1^{(n)}, X_2^{(n)}, \dots, X_n^{(n)}) = (U_1, U_2, \dots, U_n). \quad (25)$$

For $0 \leq \lambda \leq m - 1$, the intermediate vectors are defined recursively using the following relation:

$$\begin{aligned} & (X_1^{(2^\lambda)}, X_2^{(2^\lambda)}, \dots, X_n^{(2^\lambda)}) \\ & = (X_1^{(2^{\lambda+1})}, X_2^{(2^{\lambda+1})}, \dots, X_n^{(2^{\lambda+1})}) (\mathbf{I}_{2^\lambda} \otimes \mathbf{G}_2^{\text{polar}} \otimes \mathbf{I}_{2^{m-\lambda-1}}), \end{aligned} \quad (26)$$

where \mathbf{I}_n is the $n \times n$ identity matrix. By definition, $(X_1^{(1)}, X_2^{(1)}, \dots, X_n^{(1)}) = (X_1, X_2, \dots, X_n)$ is the codeword vector. Intuitively, the intermediate vector $(X_1^{(2^\lambda)}, X_2^{(2^\lambda)}, \dots, X_n^{(2^\lambda)})$ is obtained from performing $(m - \lambda)$ layers of polar transform on the message vector (U_1, U_2, \dots, U_n) . Fig. 7 gives a concrete example of the intermediate vectors in an ABS polar code, which are similar to the ones in

standard polar codes. For each $0 \leq \lambda \leq m$, $1 \leq i \leq 2^\lambda$ and $1 \leq \beta \leq 2^{m-\lambda}$, we introduce the shorthand notation

$$\begin{aligned} X_{i,\beta}^{(\lambda)} &= X_{\beta+(i-1)2^{m-\lambda}}^{(2^\lambda)}, & Y_{i,\beta}^{(\lambda)} &= Y_{\beta+(i-1)2^{m-\lambda}}, \\ \mathbf{O}_{i,\beta}^{(\lambda)} &= (X_{1,\beta}^{(\lambda)}, X_{2,\beta}^{(\lambda)}, \dots, X_{i-1,\beta}^{(\lambda)}, Y_{1,\beta}^{(\lambda)}, Y_{2,\beta}^{(\lambda)}, \dots, Y_{2^\lambda,\beta}^{(\lambda)}). \end{aligned} \quad (27)$$

According to the standard polar code construction, the $2^{m-\lambda}$ random vectors

$$\left\{ (X_{1,\beta}^{(\lambda)}, X_{2,\beta}^{(\lambda)}, \dots, X_{2^\lambda,\beta}^{(\lambda)}, Y_{1,\beta}^{(\lambda)}, Y_{2,\beta}^{(\lambda)}, \dots, Y_{2^\lambda,\beta}^{(\lambda)}) \right\}_{\beta=1}^{2^{m-\lambda}}$$

are independent and identically distributed. Moreover, the channel mapping from $X_{i,\beta}^{(\lambda)}$ to $\mathbf{O}_{i,\beta}^{(\lambda)}$ is the bit-channel $W_i^{(2^\lambda)}$ for every $1 \leq \beta \leq 2^{m-\lambda}$, where $W_i^{(2^\lambda)}$ is defined recursively using the relation (13).

Recall that (y_1, \dots, y_n) is a realization of the random vector (Y_1, \dots, Y_n) . For each $0 \leq \lambda \leq m$, $1 \leq i \leq 2^\lambda$ and $1 \leq \beta \leq 2^{m-\lambda}$, we introduce the shorthand notation $y_{i,\beta}^{(\lambda)} = y_{\beta+(i-1)2^{m-\lambda}}$, and we use $\hat{x}_{i,\beta}^{(\lambda)}$ to denote the decoded value of $X_{i,\beta}^{(\lambda)}$. Moreover, we define a vector

$$\hat{\mathbf{o}}_{i,\beta}^{(\lambda)} = (\hat{x}_{1,\beta}^{(\lambda)}, \hat{x}_{2,\beta}^{(\lambda)}, \dots, \hat{x}_{i-1,\beta}^{(\lambda)}, y_{1,\beta}^{(\lambda)}, y_{2,\beta}^{(\lambda)}, \dots, y_{2^\lambda,\beta}^{(\lambda)}). \quad (28)$$

By the analysis above, we have

$$\mathbb{P}(\mathbf{O}_{i,\beta}^{(\lambda)} = \hat{\mathbf{o}}_{i,\beta}^{(\lambda)} | X_{i,\beta}^{(\lambda)} = b) = W_i^{(2^\lambda)}(\hat{\mathbf{o}}_{i,\beta}^{(\lambda)} | b) \quad \text{for } b \in \{0, 1\}. \quad (29)$$

Now we are ready to introduce the data structures used in the SCL decoder for standard polar codes. Most of the data structures below are also used in the SCL decoder for ABS polar codes.

(i) 4-dimensional *probability array* D. The entries in the array D are indexed as

$$\begin{aligned} D[\lambda, s, \beta, b], & \quad 0 \leq \lambda \leq m, \quad 1 \leq s \leq L, \\ & \quad 1 \leq \beta \leq 2^{m-\lambda}, \quad 0 \leq b \leq 1. \end{aligned}$$

For each $0 \leq \lambda \leq m, 1 \leq s \leq L$, we define a subarray of D as

$$D[\lambda, s] = (D[\lambda, s, \beta, b], \quad 1 \leq \beta \leq 2^{m-\lambda}, \quad b \in \{0, 1\}),$$

and we use $\vec{D}[\lambda, s]$ to denote the pointer to the head address of $D[\lambda, s]$. In the algorithms below, we will write $D[\lambda, s, \beta, b]$ and $\vec{D}[\lambda, s][\beta, b]$ interchangeably. Each array $D[\lambda, s]$ is used to store a set of transition probabilities in (29).

(ii) 1-dimensional *integer array* N_D . The entries of N_D are $N_D[\lambda], 0 \leq \lambda \leq m$. The entry $N_D[\lambda]$ takes value in the set $\{0, 1, 2, \dots, L\}$ for every $0 \leq \lambda \leq m$. The value of $N_D[\lambda]$ has the following meaning: The arrays $D[\lambda, 1], D[\lambda, 2], \dots, D[\lambda, N_D[\lambda]]$ are currently occupied in the decoding procedure while the arrays $D[\lambda, N_D[\lambda] + 1], D[\lambda, N_D[\lambda] + 2], \dots, D[\lambda, L]$ are free to use. See Fig. 8 for an illustration.

(iii) 3-dimensional *bit array* B. The entries in the array B are indexed as

$$B[\lambda, s, \beta], \quad 0 \leq \lambda \leq m, \quad 1 \leq s \leq 2L, \quad 1 \leq \beta \leq 2^{m-\lambda}.$$

For each $0 \leq \lambda \leq m, 1 \leq s \leq 2L$, we define a subarray of B as

$$B[\lambda, s] = (B[\lambda, s, \beta], \quad 1 \leq \beta \leq 2^{m-\lambda}),$$

and we use $\vec{B}[\lambda, s]$ to denote the pointer to the head address of $B[\lambda, s]$. In the algorithms below, we will write $B[\lambda, s, \beta]$ and $\vec{B}[\lambda, s][\beta]$ interchangeably. Each array $B[\lambda, s]$ is used to store a set of decoding results of the intermediate vectors.

(iv) 1-dimensional *integer array* N_B . The entries of N_B are $N_B[\lambda], 0 \leq \lambda \leq m$. The entry $N_B[\lambda]$ takes value in the set $\{0, 1, 2, \dots, 2L\}$ for every $0 \leq \lambda \leq m$. The value of $N_B[\lambda]$ has the following meaning: The arrays $B[\lambda, 1], B[\lambda, 2], \dots, B[\lambda, N_B[\lambda]]$ are currently occupied in the decoding procedure while the arrays $B[\lambda, N_B[\lambda] + 1], B[\lambda, N_B[\lambda] + 2], \dots, B[\lambda, 2L]$ are free to use.

N_D	D							
$N_D[0] = 1$	D[0, 1]	D[0, 2]	D[0, 3]	D[0, 4]	D[0, 5]	D[0, 6]	D[0, 7]	D[0, 8]
$N_D[1] = 1$	D[1, 1]	D[1, 2]	D[1, 3]	D[1, 4]	D[1, 5]	D[1, 6]	D[1, 7]	D[1, 8]
$N_D[2] = 1$	D[2, 1]	D[2, 2]	D[2, 3]	D[2, 4]	D[2, 5]	D[2, 6]	D[2, 7]	D[2, 8]
$N_D[3] = 1$	D[3, 1]	D[3, 2]	D[3, 3]	D[3, 4]	D[3, 5]	D[3, 6]	D[3, 7]	D[3, 8]
$N_D[4] = 1$	D[4, 1]	D[4, 2]	D[4, 3]	D[4, 4]	D[4, 5]	D[4, 6]	D[4, 7]	D[4, 8]
$N_D[5] = 4$	D[5, 1]	D[5, 2]	D[5, 3]	D[5, 4]	D[5, 5]	D[5, 6]	D[5, 7]	D[5, 8]
$N_D[6] = 8$	D[6, 1]	D[6, 2]	D[6, 3]	D[6, 4]	D[6, 5]	D[6, 6]	D[6, 7]	D[6, 8]

Fig. 8: An illustration of D and N_D for code length $n = 64$ and list size $L = 8$. We put $D[\lambda, s]$ in a shaded cell if it is currently occupied; otherwise, we put it in a white cell. For example, $N_D[5] = 4$ means that $D[5, 1], D[5, 2], D[5, 3], D[5, 4]$ have already been allocated to store some transition probabilities while $D[5, 5], D[5, 6], D[5, 7], D[5, 8]$ are free to use.

- (v) 1-dimensional *probability array score*. The entries of `score` are `score[ℓ]`, $1 \leq \ell \leq L_c$, where $L_c \in \{1, 2, \dots, L\}$ is the current list size. Each `score[ℓ]` records the current transition probability of the ℓ th candidate in the decoding list. When the current list size is larger than the prescribed upper bound L , we prune the list according to the value of `score[ℓ]`.

- (vi) 2-dimensional *pointer arrays* P, \bar{P} . Their entries are

$$P = (P[\ell, \lambda], 1 \leq \ell \leq L, 0 \leq \lambda \leq m), \quad \bar{P} = (\bar{P}[\ell, \lambda], 1 \leq \ell \leq L, 0 \leq \lambda \leq m).$$

We use $P[\ell, \lambda]$ to store the pointer $\vec{D}[\lambda, N_D[\lambda] + 1]$, so that we can store the transition probabilities in the array $D[\lambda, N_D[\lambda] + 1]$ and access them in the future. We usually assign values (i.e., pointers) to $P[\ell, \lambda]$ through the function `allocate_prob` in Algorithm 4. The function `allocate_prob` is called in Line 4 of Algorithm 6, Line 3 of Algorithm 9, and Line 3 of Algorithm 10. The array \bar{P} is a supplement to P . We use \bar{P} when the entries in P are occupied.

- (vii) 2-dimensional *pointer arrays* R, \bar{R} . Their entries are

$$R = (R[\ell, \lambda], 1 \leq \ell \leq L, 0 \leq \lambda \leq m), \quad \bar{R} = (\bar{R}[\ell, \lambda], 1 \leq \ell \leq L, 0 \leq \lambda \leq m).$$

We use $R[\ell, \lambda]$ to store the pointer $\vec{B}[\lambda, N_B[\lambda] + 1]$, so that we can store the decoding results of intermediate vectors in the array $B[\lambda, N_B[\lambda] + 1]$ and access them in the future. We usually assign values (i.e., pointers) to $R[\ell, \lambda]$ through the function `allocate_bit` in Algorithm 5. The function `allocate_bit` is called in Line 13 of Algorithm 7 and Lines 10,16 of Algorithm 8. The array \bar{R} is a supplement to R . We use \bar{R} when the entries in R are occupied.

- (viii) *priority queue* `PriQue`. `PriQue` is a maximum priority queue with size $2L$ such that the element with the maximum value is always removed first from the queue. We use `PriQue` to record and prune candidate decoding paths. Each element in the queue is a triple (ℓ, b, prob) with the following meaning: When we decode U_i in the last layer $\lambda = m$, the (posterior) probability of $U_i = b$ in the ℓ th decoding path is `prob`. The queue `PriQue` has 4 interfaces: i) `PriQue.push(ℓ, b, prob)` pushes the element (ℓ, b, prob) to the queue; ii) `PriQue.pop()` removes the element (ℓ, b, prob) with the maximum `prob` in the queue; iii) `PriQue.clear()` removes all the remaining elements in the queue; iv) `PriQue.size()` returns the current number of elements in the queue.

We associate each candidate in the decoding list with a list element. There are at most L list elements in total. For $1 \leq \ell \leq L$, the ℓ th list element has the following fields:

$$\begin{aligned} & (\mathbf{P}[\ell, 0], \mathbf{P}[\ell, 1], \dots, \mathbf{P}[\ell, m]), \\ & (\mathbf{R}[\ell, 0], \mathbf{R}[\ell, 1], \dots, \mathbf{R}[\ell, m]), \\ & \text{score}[\ell]. \end{aligned} \tag{30}$$

Algorithm 4: `allocate_prob(λ)`

Input: layer $\lambda \in \{0, 1, 2, \dots, m\}$

Output: a pointer to the allocated memory

- 1 $N_D[\lambda] \leftarrow N_D[\lambda] + 1$
 - 2 **return** $\vec{D}[\lambda, N_D[\lambda]]$
-

Algorithm 5: `allocate_bit(λ)`

Input: layer $\lambda \in \{0, 1, 2, \dots, m\}$

Output: a pointer to the allocated memory

- 1 $N_B[\lambda] \leftarrow N_B[\lambda] + 1$
 - 2 **return** $\vec{B}[\lambda, N_B[\lambda]]$
-

The function `allocate_prob` in Algorithm 4 and the function `allocate_bit` in Algorithm 5 are used to allocate memory spaces throughout the decoding procedure. `allocate_prob(λ)` returns the pointer to the next usable array in $\mathbf{D}[\lambda, 1], \mathbf{D}[\lambda, 2], \dots, \mathbf{D}[\lambda, L]$ and updates the value of $N_D[\lambda]$. Similarly, `allocate_bit(λ)` returns the pointer to the next usable array in $\mathbf{B}[\lambda, 1], \mathbf{B}[\lambda, 2], \dots, \mathbf{B}[\lambda, 2L]$ and updates the value of $N_B[\lambda]$.

We present the main function `ST_decode((y_1, y_2, \dots, y_n))` in Algorithm 6. Note that we only update the value of the current list size in the last layer $\lambda = m$, and we have only one list element in the beginning. The first 3 lines initialize the parameters. In Line 4, we assign the pointer $\vec{D}[0, 1]$ to $\mathbf{P}[1, 0]$ and update the value of $N_D[0]$ to be 1. In Lines 5–7, we store the transition probabilities of the whole channel output vector in the array $\mathbf{D}[0, 1]$. Line 8 executes recursive decoding which we will explain later. After recursive decoding, we obtain L_c list elements. In the ℓ th list element, $\text{score}[\ell]$ is the transition probability which measures the likelihood of this list element, and the decoding result is stored in the array $(\mathbf{R}[\ell, 0][1], \mathbf{R}[\ell, 0][2], \dots, \mathbf{R}[\ell, 0][n])$. In Lines 9–17, we pick the list element with the maximum $\text{score}[\ell]$ and return the corresponding decoding result.

Before explaining the recursive decoding function `decode_channel` in Algorithm 7, let us introduce some additional notation. Recall that we defined a vector $\hat{\mathbf{o}}_{i,\beta}^{(\lambda)}$ in (28) which consists of both the decoding results of intermediate vectors and the channel outputs. This notation is designed for the SC decoder because we only have a single decoding result in the whole SC decoding procedure. However, we have multiple decoding results in the SCL decoder, so we need the following modification of the notation $\hat{\mathbf{o}}_{i,\beta}^{(\lambda)}$. For each $1 \leq \ell \leq L_c$, we use $\hat{x}_{i,\beta}^{(\ell,\lambda)}$ to denote the decoded value of $X_{i,\beta}^{(\lambda)}$ in the ℓ th list element, and we define a vector

$$\hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} = (\hat{x}_{1,\beta}^{(\ell,\lambda)}, \hat{x}_{2,\beta}^{(\ell,\lambda)}, \dots, \hat{x}_{i-1,\beta}^{(\ell,\lambda)}, y_{1,\beta}^{(\lambda)}, y_{2,\beta}^{(\lambda)}, \dots, y_{2^\lambda,\beta}^{(\lambda)}). \tag{31}$$

Then (29) becomes

$$\mathbb{P}(\mathbf{O}_{i,\beta}^{(\lambda)} = \hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} | X_{i,\beta}^{(\lambda)} = b) = W_i^{(2^\lambda)}(\hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} | b) \quad \text{for } b \in \{0, 1\}.$$

Lemma 3. *Suppose that $0 \leq \lambda \leq m$ and $1 \leq i \leq 2^\lambda$. Before we call the function `decode_channel` in Algorithm 7 with input parameters (λ, i) , the pointer $\mathbf{P}[\ell, \lambda]$ satisfies that*

$$\mathbf{P}[\ell, \lambda][\beta, b] = W_i^{(2^\lambda)}(\hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} | b) \quad \text{for all } 1 \leq \ell \leq L_c, 1 \leq \beta \leq 2^{m-\lambda} \text{ and } b \in \{0, 1\}. \tag{32}$$

Algorithm 6: ST_Decode((y_1, y_2, \dots, y_n))

Input: the received vector $(y_1, y_2, \dots, y_n) \in \mathcal{Y}^n$
Output: the decoded codeword $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n) \in \{0, 1\}^n$

- 1 **for** $\lambda \in \{1, 2, \dots, m\}$ **do**
- 2 $\lfloor N_D[\lambda] \leftarrow 0, \quad N_B[\lambda] \leftarrow 0$
- 3 $L_c \leftarrow 1$
- 4 $P[1, 0] \leftarrow \text{allocate_prob}(0)$
- 5 **for** $\beta \in \{1, 2, \dots, n\}$ **do**
- 6 **for** $b \in \{0, 1\}$ **do**
- 7 $\lfloor P[1, 0][\beta, b] \leftarrow W(y_\beta|b)$
- 8 $\text{decode_channel}(0, 1)$ ▷ Algorithm 7, recursive decoding
- 9 $\text{max_score} \leftarrow 0$
- 10 $\text{max_}\ell \leftarrow 0$
- 11 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 12 **if** $\text{score}[\ell] \geq \text{max_score}$ **then**
- 13 $\text{max_score} \leftarrow \text{score}[\ell]$
- 14 $\text{max_}\ell \leftarrow \ell$
- 15 **for** $\beta = 1, 2, \dots, n$ **do**
- 16 $\lfloor \hat{x}_\beta \leftarrow R[\text{max_}\ell, 0][\beta]$
- 17 **return** $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$

After the function $\text{decode_channel}(\lambda, i)$ in Algorithm 7 returns, the pointer $R[\ell, \lambda]$ satisfies that

$$R[\ell, \lambda][\beta] = \hat{x}_{i, \beta}^{(\ell, \lambda)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda}. \quad (33)$$

Proof. We prove (33) first, and we prove it by induction. Lines 1–2 of Algorithm 7 deal with the base case $\lambda = m$, where we decode U_i in the message vector (U_1, U_2, \dots, U_n) by calling the function $\text{decode_boundary_channel}(i)$ in Algorithm 8. By (27), when $\lambda = m$, we have $X_{i,1}^{(m)} = X_i^{(n)}$. By (25), we further obtain that $X_{i,1}^{(m)} = U_i$. If U_i is a frozen bit, then Line 17 of Algorithm 8 immediately implies (33). If U_i is an information bit, we first use $\bar{R}[\ell, m][1]$ to store the decoding result of U_i in the ℓ th list element⁴; see Line 11 of Algorithm 8. Next we swap \bar{R} and R in Line 13, so (33) is satisfied.

For the inductive step, we assume that (33) holds for $\lambda + 1$ and prove it for λ . By this induction hypothesis, after executing Line 6 of Algorithm 7, we have

$$R[\ell, \lambda + 1][\beta] = \hat{x}_{2i-1, \beta}^{(\ell, \lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

After executing Line 8 and Line 12 of Algorithm 7, we have

$$\text{temp}[\beta] = \hat{x}_{2i-1, \beta}^{(\ell, \lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

Again by the induction hypothesis, after executing Line 10, we have

$$R[\ell, \lambda + 1][\beta] = \hat{x}_{2i, \beta}^{(\ell, \lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

Since we set $n_c = 2^\lambda$ in Line 4, we have $n/(2n_c) = 2^{m-\lambda-1}$. Therefore, Lines 15–16 become

$$R[\ell, \lambda][\beta] = \hat{x}_{2i-1, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i, \beta}^{(\ell, \lambda+1)}, \quad R[\ell, \lambda][\beta + 2^{m-\lambda-1}] = \hat{x}_{2i, \beta}^{(\ell, \lambda+1)} \quad (34)$$

$$\text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

⁴The variable b in Line 11 of Algorithm 8 is the decoding result of U_i in the ℓ th list element. We will explain Algorithm 8 later.

(26)–(27) together imply that

$$X_{i,\beta}^{(\lambda)} = X_{2i-1,\beta}^{(\lambda+1)} + X_{2i,\beta}^{(\lambda+1)}, \quad X_{i,\beta+2^{m-\lambda-1}}^{(\lambda)} = X_{2i,\beta}^{(\lambda+1)} \quad \text{for all } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

This further implies that

$$\hat{x}_{i,\beta}^{(\ell,\lambda)} = \hat{x}_{2i-1,\beta}^{(\ell,\lambda+1)} + \hat{x}_{2i,\beta}^{(\ell,\lambda+1)}, \quad \hat{x}_{i,\beta+2^{m-\lambda-1}}^{(\ell,\lambda)} = \hat{x}_{2i,\beta}^{(\ell,\lambda+1)} \quad (35)$$

for all $1 \leq \ell \leq L_c$ and $1 \leq \beta \leq 2^{m-\lambda-1}$.

Combining this with (34), we complete the proof of (33).

Next we prove (32) by induction. This time the base case is $\lambda = 0$, and this case only occurs once in Line 8 of Algorithm 6 during the whole decoding procedure. Note that the channel $W_1^{(1)}$ is W itself. Therefore, Lines 5–7 of Algorithm 6 immediately imply (32) for $\lambda = 0$.

For the inductive step, we assume that (32) holds for λ and prove it for $\lambda + 1$. By this induction hypothesis, (32) holds for λ when we execute Line 5 of Algorithm 7. In other words, the array associated with the pointer $P[\ell, \lambda]$ stores the transition probabilities of $W_i^{(2^\lambda)}$. By (13), $W_{2i-1}^{(2^{\lambda+1})}$ is the “−” transform of $W_i^{(2^\lambda)}$. The function `calculate_−_transform`($\lambda + 1$) calculates the “−” transform of $W_i^{(2^\lambda)}$ and stores the results in the array associated with the pointer $P[\ell, \lambda + 1]$, so (32) holds before we call `decode_channel` in Line 6 of Algorithm 7. Again by (13), $W_{2i}^{(2^{\lambda+1})}$ is the “+” transform of $W_i^{(2^\lambda)}$. The function `calculate_+_transform`($\lambda + 1$) in Line 9 of Algorithm 7 calculates the “+” transform of $W_i^{(2^\lambda)}$ and stores the results in the array associated with the pointer $P[\ell, \lambda + 1]$, so (32) holds before we call `decode_channel` in Line 10 of Algorithm 7.

During the whole decoding procedure, the function `decode_channel` is only called in Line 8 of Algorithm 6 and Lines 6,10 of Algorithm 7. We have proved that (32) holds for all three places. This completes the proof of the lemma. \square

Now let us explain how Algorithm 8 works when U_i is an information bit. First, we explore both cases $U_i = 0$ and $U_i = 1$ for every list element; see Lines 2–4. The variable b in Lines 3–4 represents the (possible) value of U_i . Since we explore two possible paths for each existing list element, we have expanded the list size by a factor of 2 after executing Lines 2–4. If the current list size is larger than L , then we need to prune the list, and this is done in Lines 5–13. In Line 5, we update the current list size L_c to be the smaller value among L and the size of `PriQue`. Then in Lines 6–11, we execute `PriQue.pop()` L_c times to obtain L_c elements in the queue with the largest value of `score`[ℓ]. By Line 4, `score`[ℓ] stores the transition probability $P[\ell, m][1, b]$, which measures the likelihood of the ℓ th list element. Therefore, we obtain L_c list elements with the largest likelihood after executing Lines 6–11.

The next lemma shows that the data structures D and B are large enough to store the transition probabilities and the decoding results of the intermediate vectors throughout the decoding procedure.

Lemma 4. *Throughout the whole decoding procedure, we have $N_D[\lambda] \leq L$ and $N_B[\lambda] \leq 2L$ for all $0 \leq \lambda \leq m$. The space complexity of the SCL decoder is $O(Ln)$.*

Proof. For every $0 \leq \lambda \leq m$ and every $1 \leq i \leq 2^\lambda$, the function `decode_channel`(λ, i) is called only once. Moreover, the function `decode_channel`($\lambda, i + 1$) is always called after the function `decode_channel`(λ, i) returns. Each time we call the function `decode_channel`(λ, i), we only need to store the transition probabilities for $L_c \leq L$ different decoding paths, and we always reset $N_D[\lambda]$ to 0 before the function `decode_channel`(λ, i) returns, so $N_D[\lambda] \leq L$.

We need to store the decoding results of intermediate vectors for $L_c \leq L$ list elements when we call the function `decode_channel`($\lambda + 1, 2i - 1$) in Line 6 of Algorithm 7. Similarly, we need to store the decoding results of intermediate vectors for another $L'_c \leq L$ list elements⁵ when we call the function `decode_channel`($\lambda + 1, 2i$) in Line 10 of Algorithm 7. Therefore, before we reset $N_B[\lambda + 1]$ to 0 in Line 17, we have $N_B[\lambda + 1] = L_c + L'_c \leq 2L$. This proves that $N_B[\lambda]$ can not exceed $2L$ for all $0 \leq \lambda \leq m$.

⁵We use L'_c here because the current list size may change over the decoding procedure.

Algorithm 7: decode_channel(λ, i)

Input: layer $\lambda \in \{0, 1, 2, \dots, m\}$ and index $i \in \{1, 2, \dots, 2^\lambda\}$

```

1 if  $\lambda = m$  then
2   | decode_boundary_channel( $i$ )                                ▷ Algorithm 8
3 else
4   |  $n_c \leftarrow 2^\lambda$                                        ▷  $W_{2i-1}^{(2n_c)} = (W_i^{(n_c)})_-$ 
5   | calculate_-_transform( $\lambda + 1$ )
6   | decode_channel( $\lambda + 1, 2i - 1$ )
7   | for  $\ell \in \{1, 2, \dots, L_c\}$  do
8   |   |  $R[\ell, \lambda] \leftarrow R[\ell, \lambda + 1]$ 
9   |   | calculate_+_transform( $\lambda + 1$ )                             ▷  $W_{2i}^{(2n_c)} = (W_i^{(n_c)})_+$ 
10  |   | decode_channel( $\lambda + 1, 2i$ )
11  |   | for  $\ell \in \{1, 2, \dots, L_c\}$  do
12  |   |   | temp  $\leftarrow R[\ell, \lambda]$ 
13  |   |   |  $R[\ell, \lambda] \leftarrow \text{allocate\_bit}(\lambda)$ 
14  |   |   | for  $\beta \in \{1, 2, \dots, n/(2n_c)\}$  do
15  |   |   |   |  $R[\ell, \lambda][\beta] \leftarrow \text{temp}[\beta] + R[\ell, \lambda + 1][\beta]$ 
16  |   |   |   |  $R[\ell, \lambda][\beta + n/(2n_c)] \leftarrow R[\ell, \lambda + 1][\beta]$ 
17  |   |  $N_B[\lambda + 1] \leftarrow 0$ 
18  |  $N_D[\lambda] \leftarrow 0$ 
19 return

```

Next we prove the $O(Ln)$ space complexity of the SCL decoder. The number of entries in the array D is upper bounded by

$$2L \sum_{\lambda=0}^m 2^{m-\lambda} = 2L(1 + 2 + 4 + \dots + 2^m) < 2L \cdot 2^{m+1} = 4Ln.$$

Similarly, the number of entries in B is upper bounded by

$$2L \sum_{\lambda=0}^m 2^{m-\lambda} < 4Ln.$$

The number of entries in both N_D and N_B is $O(\log(n))$. The number of entries in both score and PriQue is $O(L)$. The number of entries in the pointer arrays P, \bar{P}, R, \bar{R} is $O(L \log(n))$. Adding these up gives us the $O(Ln)$ space complexity. \square

Proposition 2. *The decoding time complexity of standard polar codes is $O(Ln \log(n))$.*

B. SCL decoder for standard polar codes based on the Double-Bits polar transform

In this subsection, we present a new SCL decoder for standard polar codes based on the Double-Bits polar transform in Fig. 3. We still use the notation in (25)–(27) and (31). By (15), the channel mapping from $(X_{i,\beta}^{(\lambda)}, X_{i+1,\beta}^{(\lambda)})$ to $\mathbf{O}_{i,\beta}^{(\lambda)}$ is the adjacent-bits-channel $V_i^{(2^\lambda)}$ for every $1 \leq \beta \leq 2^{m-\lambda}$, i.e.,

$$\mathbb{P}(\mathbf{O}_{i,\beta}^{(\lambda)} = \hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} | X_{i,\beta}^{(\lambda)} = a, X_{i+1,\beta}^{(\lambda)} = b) = V_i^{(2^\lambda)}(\hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} | a, b) \quad \text{for } a, b \in \{0, 1\}. \quad (36)$$

Below we list the data structures of the new SCL decoder for standard polar codes based on the DB polar transform.

Algorithm 8: decode_boundary_channel(i)

Input: index i in the last layer ($\lambda = m$)

1 **if** $i \in \mathcal{A}$ **then** $\triangleright U_i$ is an information bit

2 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**

3 **for** $b \in \{0, 1\}$ **do**

4 $\text{PriQue.push}(\ell, b, \text{P}[\ell, m][1, b])$

5 $L_c \leftarrow \min\{L, \text{PriQue.size}()\}$

6 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**

7 $(\ell', b, \text{score}[\ell]) \leftarrow \text{PriQue.pop}()$

8 **for** $\lambda \in \{0, 1, 2, \dots, m-1\}$ **do**

9 $(\bar{\text{P}}[\ell, \lambda], \bar{\text{R}}[\ell, \lambda]) \leftarrow (\text{P}[\ell', \lambda], \text{R}[\ell', \lambda])$

10 $\bar{\text{R}}[\ell, m] \leftarrow \text{allocate_bit}(m)$

11 $\bar{\text{R}}[\ell, m][1] \leftarrow b$

12 $\text{PriQue.clear}()$ \triangleright Remove all the remaining elements

13 $\text{swap}(\bar{\text{P}}, \text{P}), \text{swap}(\bar{\text{R}}, \text{R})$

14 **else** $\triangleright U_i$ is a frozen bit

15 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**

16 $\text{R}[\ell, m] \leftarrow \text{allocate_bit}(m)$

17 $\text{R}[\ell, m][1] \leftarrow$ frozen value of U_i

18 **return**

Algorithm 9: calculate_--_transform(λ)

Input: layer $1 \leq \lambda \leq m$

Output: Update the entries pointed by $\text{P}[\ell, \lambda]$, $1 \leq \ell \leq L_c$

1 $\bar{n}_c \leftarrow 2^{m-\lambda}$

2 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**

3 $\text{P}[\ell, \lambda] \leftarrow \text{allocate_prob}(\lambda)$

4 **for** $\beta \in \{1, 2, \dots, \bar{n}_c\}, a \in \{0, 1\}$ **do**

5 $\beta' \leftarrow \beta + \bar{n}_c$

6 $\text{P}[\ell, \lambda][\beta, a] \leftarrow \frac{1}{2} \sum_{b \in \{0, 1\}} \text{P}[\ell, \lambda-1][\beta, a+b] \text{P}[\ell, \lambda-1][\beta', b]$

7 **return**

(i) 5-dimensional *probability array* D . The entries in the array D are indexed as

$$\begin{aligned} \text{D}[\lambda, s, \beta, a, b], \quad 1 \leq \lambda \leq m, \quad 1 \leq s \leq L, \\ 1 \leq \beta \leq 2^{m-\lambda}, \quad 0 \leq a, b \leq 1. \end{aligned}$$

For each $1 \leq \lambda \leq m, 1 \leq s \leq L$, we define a subarray of D as

$$\text{D}[\lambda, s] = (\text{D}[\lambda, s, \beta, a, b], \quad 1 \leq \beta \leq 2^{m-\lambda}, \quad 0 \leq a, b \leq 1),$$

and we use $\vec{\text{D}}[\lambda, s]$ to denote the pointer to the head address of $\text{D}[\lambda, s]$. In the algorithms below, we will write $\text{D}[\lambda, s, \beta, a, b]$ and $\vec{\text{D}}[\lambda, s][\beta, a, b]$ interchangeably. Each array $\text{D}[\lambda, s]$ is used to store a set of transition probabilities in (36).

(ii) 1-dimensional *integer array* N_D . The entries of N_D are $\text{N}_\text{D}[\lambda], 1 \leq \lambda \leq m$. This array is defined in the same way as the previous subsection.

Algorithm 10: calculate_+_transform(λ)

Input: layer $1 \leq \lambda \leq m$
Output: Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

```

1  $\bar{n}_c \leftarrow 2^{m-\lambda}$ 
2 for  $\ell \in \{1, 2, \dots, L_c\}$  do
3    $P[\ell, \lambda] \leftarrow \text{allocate\_prob}(\lambda)$ 
4   for  $\beta \in \{1, 2, \dots, \bar{n}_c\}, b \in \{0, 1\}$  do
5      $a \leftarrow R[\ell, \lambda - 1][\beta]$ 
6      $\beta' \leftarrow \beta + \bar{n}_c$ 
7      $P[\ell, \lambda][\beta, b] \leftarrow \frac{1}{2}P[\ell, \lambda - 1][\beta, a + b]P[\ell, \lambda - 1][\beta', b]$ 
8 return

```

(iii) 3-dimensional *bit array* B . The entries in the array B are indexed as

$$B[\lambda, s, \beta], \quad 1 \leq \lambda \leq m, \quad 1 \leq s \leq 4L, \quad 1 \leq \beta \leq 2^{m-\lambda}. \quad (37)$$

For each $1 \leq \lambda \leq m, 1 \leq s \leq 4L$, we define a subarray of B as

$$B[\lambda, s] = (B[\lambda, s, \beta], \quad 1 \leq \beta \leq 2^{m-\lambda}),$$

and we use $\vec{B}[\lambda, s]$ to denote the pointer to the head address of $B[\lambda, s]$. In the algorithms below, we will write $B[\lambda, s, \beta]$ and $\vec{B}[\lambda, s][\beta]$ interchangeably. Each array $B[\lambda, s]$ is used to store a set of decoding results of the intermediate vectors.

- (iv) 1-dimensional *integer array* N_B . The entries of N_B are $N_B[\lambda]$, $1 \leq \lambda \leq m$. The entry $N_B[\lambda]$ takes value in the set $\{0, 1, 2, \dots, 4L\}$ for every $1 \leq \lambda \leq m$. The meaning of $N_B[\lambda]$ is the same as the previous subsection.
- (v) 1-dimensional *probability array score*, defined in the same way as the previous subsection.
- (vi) 2-dimensional *pointer arrays* P, \bar{P} . Their entries are

$$P = (P[\ell, \lambda], \quad 1 \leq \ell \leq L, \quad 1 \leq \lambda \leq m), \quad \bar{P} = (\bar{P}[\ell, \lambda], \quad 1 \leq \ell \leq L, \quad 1 \leq \lambda \leq m).$$

They are used in the same way as the previous subsection.

- (vii) 2-dimensional *pointer arrays* R, \bar{R} . Their entries are

$$R = (R[\ell, \lambda], \quad 1 \leq \ell \leq L, \quad 1 \leq \lambda \leq m), \quad \bar{R} = (\bar{R}[\ell, \lambda], \quad 1 \leq \ell \leq L, \quad 1 \leq \lambda \leq m).$$

They are used in the same way as the previous subsection.

- (viii) 2-dimensional *pointer arrays* H, \bar{H} . Their entries are

$$H = (H[\ell, \lambda], \quad 1 \leq \ell \leq L, \quad 1 \leq \lambda \leq m), \quad \bar{H} = (\bar{H}[\ell, \lambda], \quad 1 \leq \ell \leq L, \quad 1 \leq \lambda \leq m).$$

These two pointer arrays serve as backups of R and \bar{R} . We use H, \bar{H} when all the entries in R and \bar{R} are occupied.

- (ix) *priority queue PriQue*. PriQue is defined essentially in the same way as the previous subsection. The only difference is that each element in the queue changes from a triple (ℓ, b, prob) to a quadruple $(\ell, a, b, \text{prob})$. The quadruple $(\ell, a, b, \text{prob})$ has the following meaning: When we decode U_i and U_{i+1} in the last layer $\lambda = m$, the (posterior) probability of $(U_i = a, U_{i+1} = b)$ in the ℓ th decoding path is prob .

Below we list the main differences between the data structures in this subsection and the previous subsection.

- (1) The range of λ in all the data structures changes from $0 \leq \lambda \leq m$ (previous subsection) to $1 \leq \lambda \leq m$ (this subsection).

- (2) The dimension of the probability array D changes from 4 (previous subsection) to 5 (this subsection).
- (3) The range of the index s in the array B changes from $1 \leq s \leq 2L$ (previous subsection) to $1 \leq s \leq 4L$ (this subsection).
- (4) We have two more pointer arrays H, \bar{H} in this subsection.
- (5) Each element in the priority queue `PriQue` changes from a triple (ℓ, b, prob) to a quadruple $(\ell, a, b, \text{prob})$.

For the SCL decoder presented in this subsection, the ℓ th list element has the following fields:

$$\begin{aligned}
 & (P[\ell, 1], P[\ell, 2], \dots, P[\ell, m]), \\
 & (R[\ell, 1], R[\ell, 2], \dots, R[\ell, m]), \\
 & (H[\ell, 1], H[\ell, 2], \dots, H[\ell, m]), \\
 & \text{score}[\ell].
 \end{aligned} \tag{38}$$

We still use the function `allocate_prob`(λ) in Algorithm 4 although the range of λ is $\{1, 2, \dots, m\}$ in this subsection. However, we will use the function `allocate_bit` in Algorithm 11 for the new decoder in this subsection, which is different from the function with the same name in Algorithm 5. The main difference is that the function `allocate_bit` in Algorithm 11 has an extra input parameter k , which takes value in $\{1, 2\}$. In this subsection, the decoder makes decisions according to the transition probabilities of adjacent-bits-channels. Each adjacent-bits-channel has two input bits. In some cases we only decode one bit while in other cases we need to decode both bits. The input parameter k in Algorithm 11 corresponds to the number of input bits we need to decode for each adjacent-bits-channel. We do not have the parameter k in Algorithm 5 because each bit-channel only has one input bit.

Algorithm 11: `allocate_bit`(λ, k)

Input: layer $\lambda \in \{1, 2, \dots, m\}$ and an integer $k \in \{1, 2\}$

Output: a pointer to the allocated memory

- 1 $s \leftarrow N_B[\lambda] + 1$
 - 2 $N_B[\lambda] \leftarrow N_B[\lambda] + k$
 - 3 **return** $\vec{B}[\lambda, s]$
-

We present the main function `decode` in Algorithm 12. The first 3 lines initialize the parameters. In Line 4, we assign the pointer $\vec{D}[1, 1]$ to $P[1, 1]$ and update the value of $N_D[1]$ to be 1. In Lines 5–7, we calculate the transition probabilities $V_1^{(2)}(y_\beta, y_{\beta+n/2}|a, b)$ for $1 \leq \beta \leq n/2$ and $a, b \in \{0, 1\}$ using (18) and store $V_1^{(2)}(y_\beta, y_{\beta+n/2}|a, b)$ in $P[1, 1][\beta, a, b]$. Line 8 executes recursive decoding which we will explain later. After recursive decoding, we obtain L_c list elements. In the ℓ th list element, $\text{score}[\ell]$ is the transition probability which measures the likelihood of this list element. In Lines 9–14, we pick the list element with the maximum $\text{score}[\ell]$. Recall that $\hat{x}_{i,\beta}^{(\ell,\lambda)}$ is the decoded value of $X_{i,\beta}^{(\lambda)}$ in the ℓ th list element. As we will prove in Lemma 5 below, after recursive decoding, we have

$$R[\ell, 1][\beta] = \hat{x}_{1,\beta}^{(\ell,1)}, \quad R[\ell, 1][\beta + n/2] = \hat{x}_{2,\beta}^{(\ell,1)} \quad \text{for } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq n/2.$$

Since the codeword vector (X_1, \dots, X_n) and the intermediate vectors $(X_{1,1}^{(1)}, \dots, X_{1,n/2}^{(1)}), (X_{2,1}^{(1)}, \dots, X_{2,n/2}^{(1)})$ satisfy

$$X_\beta = X_{1,\beta}^{(1)} + X_{2,\beta}^{(1)}, \quad X_{\beta+n/2} = X_{2,\beta}^{(1)} \quad \text{for } 1 \leq \beta \leq n/2,$$

we further have

$$\hat{x}_\beta^{(\ell)} = R[\ell, 1][\beta] + R[\ell, 1][\beta + n/2], \quad \hat{x}_{\beta+n/2}^{(\ell)} = R[\ell, 1][\beta + n/2] \quad \text{for } 1 \leq \beta \leq n/2,$$

where $(\hat{x}_1^{(\ell)}, \hat{x}_2^{(\ell)}, \dots, \hat{x}_n^{(\ell)})$ is the decoding result of the codeword vector in the ℓ th list element. This is how we calculate the final decoding result in Lines 15–17.

Algorithm 12: Decode((y_1, y_2, \dots, y_n))

Input: the received vector $(y_1, y_2, \dots, y_n) \in \mathcal{Y}^n$
Output: the decoded codeword $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n) \in \{0, 1\}^n$

```

1 for  $\lambda \in \{1, 2, \dots, m\}$  do
2    $N_D[\lambda] \leftarrow N_B[\lambda] \leftarrow 0$ 
3  $L_c \leftarrow 1$ 
4  $P[1, 1] \leftarrow \text{allocate\_prob}(1)$ 
5 for  $\beta \in \{1, 2, \dots, n/2\}$  do
6   for  $a \in \{0, 1\}, b \in \{0, 1\}$  do
7      $P[1, 1][\beta, a, b] \leftarrow W(y_\beta | a + b) \cdot W(y_{\beta+n/2} | b)$ 
8  $\text{decode\_channel}(1, 1)$  ▷ Recursive decoding
9  $\text{max\_score} \leftarrow 0$ 
10  $\text{max\_}\ell \leftarrow 0$ 
11 for  $\ell \in \{1, 2, \dots, L_c\}$  do
12   if  $\text{score}[\ell] \geq \text{max\_score}$  then
13      $\text{max\_score} \leftarrow \text{score}[\ell]$ 
14      $\text{max\_}\ell \leftarrow \ell$ 
15 for  $\beta = 1, 2, \dots, n/2$  do
16    $\hat{x}_\beta \leftarrow R[\text{max\_}\ell, 1][\beta] + R[\text{max\_}\ell, 1][\beta + n/2]$ 
17    $\hat{x}_{\beta+n/2} \leftarrow R[\text{max\_}\ell, 1][\beta + n/2]$ 
18 return  $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$ 

```

The recursive decoding function `decode_channel` in Algorithm 13 has two branches. If $\lambda = m$, we call the function `decode_boundary_channel` in Algorithm 17. If $\lambda < m$, we call the function `decode_original_channel` in Algorithm 18. In Algorithms 17–18, we only decode $(X_{i,\beta}^{(\lambda)}, 1 \leq \beta \leq 2^{m-\lambda})$ if $i \leq 2^\lambda - 2$; we decode both $(X_{i,\beta}^{(\lambda)}, 1 \leq \beta \leq 2^{m-\lambda})$ and $(X_{i+1,\beta}^{(\lambda)}, 1 \leq \beta \leq 2^{m-\lambda})$ if $i = 2^\lambda - 1$. The following lemma further explains how Algorithms 13, 17–18 work.

Lemma 5. *Suppose that $1 \leq \lambda \leq m$ and $1 \leq i \leq 2^\lambda - 1$. Before we call the function `decode_channel` in Algorithm 13 with input parameters (λ, i) , the pointer $P[\ell, \lambda]$ satisfies that*

$$P[\ell, \lambda][\beta, a, b] = V_i^{(2^\lambda)}(\hat{\theta}_{i,\beta}^{(\ell,\lambda)} | a, b) \quad \text{for all } 1 \leq \ell \leq L_c, 1 \leq \beta \leq 2^{m-\lambda} \text{ and } a, b \in \{0, 1\}. \quad (39)$$

After the function `decode_channel`(λ, i) in Algorithm 13 returns, the pointer $R[\ell, \lambda]$ satisfies that

$$R[\ell, \lambda][\beta] = \hat{x}_{i,\beta}^{(\ell,\lambda)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda}. \quad (40)$$

Moreover, if $i = 2^\lambda - 1$, then the pointer $R[\ell, \lambda]$ further satisfies that

$$R[\ell, \lambda][\beta + 2^{m-\lambda}] = \hat{x}_{i+1,\beta}^{(\ell,\lambda)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda}. \quad (41)$$

Proof. We first prove (40)–(41) by induction. Algorithm 17 deals with the base case $\lambda = m$. Recall from (27) that $X_{i,1}^{(m)} = X_i^{(n)}$ and $X_{i+1,1}^{(m)} = X_{i+1}^{(n)}$ when $\lambda = m$. By (25), we further obtain $X_{i,1}^{(m)} = U_i$ and $X_{i+1,1}^{(m)} = U_{i+1}$. The discussion below is divided into two cases. **Case (1)** $i \leq n - 2$: If U_i is a frozen bit, then Line 11 of Algorithm 17 immediately implies (40). If U_i is an information bit, then we explore both decoding paths $U_i = 0$ and $U_i = 1$ for every list element, where the variable a in Lines 4–6 represents the (possible) value of U_i . The question mark “?” in Line 6 means that we do not need to decode U_{i+1} when $i \leq n - 2$. Since we expand the current list size by a factor of 2 in Lines 4–6, the current list

size might exceed the prescribed upper bound L . In this case, we prune the list according to $\text{score}[\ell]$ in Lines 29–44. The variables a and b in Lines 29–44 represent the decoded values of U_i and U_{i+1} in each list element, respectively. In Line 42, we use $\bar{\mathbf{R}}[\ell, m][1]$ to temporarily store the decoding result of U_i in the ℓ th list element, and we use $\bar{\mathbf{R}}[\ell, m][2]$ to temporarily store the decoding result of U_{i+1} in the ℓ th list element. Next we swap $\bar{\mathbf{R}}$ and \mathbf{R} in Line 44, so (40)–(41) are satisfied. **Case (2)** $i = n - 1$: This case is handled in Lines 12–28. Note that U_{n-1} and U_n in Lines 17, 21, 25 refer to their frozen values (or true values). The argument for Case (2) is similar to Case (1), and we do not repeat it here.

For the inductive step, we assume that (40)–(41) hold for $\lambda + 1$ and prove them for λ . By this induction hypothesis, after executing Lines 1–5 of Algorithm 18, we have

$$\mathbf{H}[\ell, \lambda + 1][\beta] = \hat{x}_{2i-1, \beta}^{(\ell, \lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

In Line 1, we set $n_c = 2^\lambda$. We again divide the discussion into two cases. **Case (1)** $i \leq n_c - 2$: In this case, we only need to prove (40). After executing Lines 7–10 and Line 14, we have

$$\text{temp}[\beta] = \hat{x}_{2i, \beta}^{(\ell, \lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

Since $n_c = 2^\lambda$, we have $n/(2n_c) = 2^{m-\lambda-1}$. Therefore, Lines 18–19 of Algorithm 18 become (34). Combining (34) with (35), we finish the proof of (40) for Case (1). **Case (2)** $i = n_c - 1$: In this case, we need to prove both (40) and (41). The proof of (40) is exactly the same as Case (1). To prove (41), we observe that if $i = n_c - 1$, then $2i + 1 = 2n_c - 1 = 2^{\lambda+1} - 1$. Then by the induction hypothesis, after executing Line 24, we have

$$\begin{aligned} \mathbf{R}[\ell, \lambda + 1][\beta] &= \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)}, & \mathbf{R}[\ell, \lambda + 1][\beta + 2^{m-\lambda-1}] &= \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)} \\ &\text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned}$$

Therefore, Lines 32–34 become

$$\begin{aligned} \mathbf{R}[\ell, \lambda][\beta + 2^{m-\lambda}] &= \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)}, & \mathbf{R}[\ell, \lambda][\beta + 2^{m-\lambda-1} + 2^{m-\lambda}] &= \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)} \\ &\text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned} \tag{42}$$

Replacing i with $i + 1$ in (35) we obtain

$$\begin{aligned} \hat{x}_{i+1, \beta}^{(\ell, \lambda)} &= \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)}, & \hat{x}_{i+1, \beta+2^{m-\lambda-1}}^{(\ell, \lambda)} &= \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)} \\ &\text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned}$$

Combining this with (42), we complete the proof of (41).

Next we prove (39) by induction. This time the base case is $\lambda = 1$, and this case only occurs once in Line 8 of Algorithm 12 during the whole decoding procedure. By (18), we have $V_1^{(2)}(y_\beta, y_{\beta+n/2}|a, b) = W(y_\beta|a+b) \cdot W(y_{\beta+n/2}|b)$. Therefore, Lines 5–7 of Algorithm 12 immediately imply (39) for $\lambda = 1$.

For the inductive step, we assume that (39) holds for λ and prove it for $\lambda + 1$. By this induction hypothesis, (39) holds for λ when we execute Line 2 of Algorithm 18. In other words, the array associated with the pointer $\mathbf{P}[\ell, \lambda]$ stores the transition probabilities of $V_i^{(2^\lambda)}$. By Lemma 1, $V_{2i-1}^{(2^{\lambda+1})}$ is the “ ∇ ” transform of $V_i^{(2^\lambda)}$. The function $\text{calculate_}\nabla\text{_transform}(\lambda + 1)$ calculates the “ ∇ ” transform of $V_i^{(2^\lambda)}$ and stores the results in the array associated with the pointer $\mathbf{P}[\ell, \lambda + 1]$, so (39) holds before we call decode_channel in Line 3 of Algorithm 18. Again by Lemma 1, $V_{2i}^{(2^{\lambda+1})}$ is the “ \diamond ” transform of $V_i^{(2^\lambda)}$. The function $\text{calculate_}\diamond\text{_transform}(\lambda + 1)$ in Line 7 of Algorithm 18 calculates the “ \diamond ” transform of $V_i^{(2^\lambda)}$ and stores the results in the array associated with the pointer $\mathbf{P}[\ell, \lambda + 1]$, so (39) holds before we call decode_channel in Line 8 of Algorithm 18. Using exactly the same method, we can show that (39) also holds before we call decode_channel in Line 24 of Algorithm 18.

Algorithm 13: decode_channel(λ, i)

Input: layer $\lambda \in \{1, 2, \dots, m\}$ and index $i \in \{1, 2, \dots, 2^\lambda - 1\}$

- 1 **if** $\lambda = m$ **then**
- 2 | decode_boundary_channel(i) ▷ Algorithm 17
- 3 **else**
- 4 | decode_original_channel(λ, i) ▷ Algorithm 18
- 5 $N_D[\lambda] \leftarrow 0$
- 6 **return**

During the whole decoding procedure, the function decode_channel is only called in Line 8 of Algorithm 12 and Lines 3,8,24 of Algorithm 18. We have proved that (39) holds for all four places. This completes the proof of the lemma. \square

Before proceeding further, let us explain the meaning of the boolean variable “flag” in Algorithm 17. flag takes value 0 if we do not expand the decoding list in the decoding procedure, and it takes value 1 otherwise. In Algorithm 17, we do not expand the decoding list if and only if we only decode frozen bits. There are two such cases, one in Lines 7–11 and the other in Lines 13–17. We set the variable flag to be 0 in both cases. In all the other cases, we need to decode at least one information bit, and we need to expand the list size by a factor of at least 2, so we set the variable flag to be 1 in all the other cases. If flag= 0, then the list size does not change, and we do not need to prune the list. Therefore, we only prune the list when flag= 1; see Line 29.

Remark 1. *The calculations in Line 6 of Algorithm 14 correspond to the “ ∇ ” transform in Fig. 3 and the first equation in (14). The calculations in Line 7 of Algorithm 15 correspond to the “ \diamond ” transform in Fig. 3 and the second equation in (14). The calculations in Lines 7-8 of Algorithm 16 correspond to the “ Δ ” transform in Fig. 3 and the third equation in (14). This is why we say that the SCL decoder presented in this subsection is based on the DB polar transform.*

Algorithm 14: calculate_ ∇ _transform(λ)

Input: layer $2 \leq \lambda \leq m$

Output: Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

- 1 $\bar{n}_c \leftarrow 2^{m-\lambda}$
- 2 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 3 | $P[\ell, \lambda] \leftarrow \text{allocate_prob}(\lambda)$
- 4 | **for** $\beta \in \{1, 2, \dots, \bar{n}_c\}$, $r_1, r_2 \in \{0, 1\}$ **do**
- 5 | | $\beta' \leftarrow \beta + \bar{n}_c$
- 6 | | $P[\ell, \lambda][\beta, r_1, r_2] \leftarrow \frac{1}{4} \sum_{r_3, r_4 \in \{0, 1\}} P[\ell, \lambda - 1][\beta, r_1 + r_2, r_3 + r_4] P[\ell, \lambda - 1][\beta', r_2, r_4]$
- 7 **return**

The next lemma shows that the data structures D and B are large enough to store the transition probabilities and the decoding results of the intermediate vectors throughout the decoding procedure.

Lemma 6. *Throughout the whole decoding procedure, we have $N_D[\lambda] \leq L$ and $N_B[\lambda] \leq 4L$ for all $1 \leq \lambda \leq m$. The space complexity of the SCL decoder is $O(Ln)$.*

Proof. The proof of $N_D[\lambda] \leq L$ is the same as Lemma 4, and we do not repeat it. Now we prove $N_B[\lambda] \leq 4L$. As we can see from Algorithms 13,17–18, each time we call the function decode_channel(λ, i), the value of $N_B[\lambda]$ increases by L_c if $i \leq 2^\lambda - 2$, and it increases by $2L_c$ if $i = 2^\lambda - 1$. Since the input

Algorithm 15: calculate $_{\diamond}$ _transform(λ)

Input: layer $2 \leq \lambda \leq m$
Output: Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

- 1 $\bar{n}_c \leftarrow 2^{m-\lambda}$
- 2 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 3 $P[\ell, \lambda] \leftarrow \text{allocate_prob}(\lambda)$
- 4 **for** $\beta \in \{1, 2, \dots, \bar{n}_c\}, r_2, r_3 \in \{0, 1\}$ **do**
- 5 $r_1 \leftarrow H[\ell, \lambda][\beta]$
- 6 $\beta' \leftarrow \beta + \bar{n}_c$
- 7 $P[\ell, \lambda][\beta, r_2, r_3] \leftarrow \frac{1}{4} \sum_{r_4 \in \{0, 1\}} P[\ell, \lambda - 1][\beta, r_1 + r_2, r_3 + r_4] P[\ell, \lambda - 1][\beta', r_2, r_4]$
- 8 **return**

Algorithm 16: calculate $_{\Delta}$ _transform(λ)

Input: layer $2 \leq \lambda \leq m$
Output: Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

- 1 $\bar{n}_c \leftarrow 2^{m-\lambda}$
- 2 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 3 $P[\ell, \lambda] \leftarrow \text{allocate_prob}(\lambda)$
- 4 **for** $\beta \in \{1, 2, \dots, \bar{n}_c\}, r_3, r_4 \in \{0, 1\}$ **do**
- 5 $r_1 \leftarrow H[\ell, \lambda][\beta], r_2 \leftarrow R[\ell, \lambda - 1][\beta]$
- 6 $\beta' \leftarrow \beta + \bar{n}_c$
- 7 $P[\ell, \lambda][\beta, r_3, r_4] \leftarrow$
- 8 $\frac{1}{4} P[\ell, \lambda - 1][\beta, r_1 + r_2, r_3 + r_4] P[\ell, \lambda - 1][\beta', r_2, r_4]$
- 9 **return**

i in Algorithm 18 satisfies $i \leq 2^\lambda - 1$, we have $2i - 1 < 2i \leq 2^{\lambda+1} - 2$. Therefore, after executing Line 3 of Algorithm 18, the value of $N_B[\lambda + 1]$ increases by at most L . Similarly, after executing Line 8 of Algorithm 18, the value of $N_B[\lambda + 1]$ also increases by at most L . If $i = 2^\lambda - 1$, we will execute Line 24 of Algorithm 18. In this case, $2i + 1 = 2^{\lambda+1} - 1$, so the value of $N_B[\lambda + 1]$ increases by at most $2L$. Therefore, before we reset $N_B[\lambda + 1]$ to 0 in Line 35 of Algorithm 18, its value is at most $4L$. This proves $N_B[\lambda] \leq 4L$.

The proof of the space complexity is the same as Lemma 4. \square

In TABLE I, we list the upper bound of $N_B[\lambda + 1]$ at the starting point and the end of the function $\text{decode_channel}(\lambda + 1, j)$. The starting point refers to the moment we call $\text{decode_channel}(\lambda + 1, j)$, and the end refers to the moment this function returns. These upper bounds come from the proof of Lemma 6.

cases		start	end
$1 \leq i \leq 2^\lambda - 1$	$j = 2i - 1$	0	L
	$j = 2i$	L	2L
$i = 2^\lambda - 1$	$j = 2i + 1$	2L	4L

TABLE I: The upper bound of $N_B[\lambda + 1]$ at the starting point and the end of the function $\text{decode_channel}(\lambda + 1, j)$

Proposition 3. *The decoding time complexity of standard polar codes based on the DB polar transform is $O(Ln \log(n))$.*

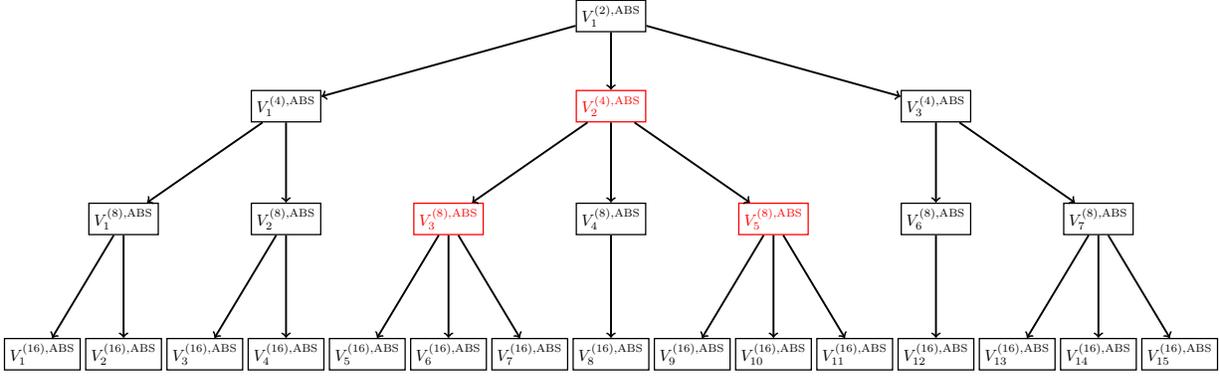


Fig. 9: Recursive decoding of the ABS polar code defined in Fig. 7. We put $V_i^{(2^\lambda),\text{ABS}}$ in a black block (e.g., $V_1^{(2),\text{ABS}}$) if $2i \notin \mathcal{I}^{(2^{\lambda+1})}$. In this case, `decode_channel`(λ, i) in Algorithm 19 calls `decode_original_channel`(λ, i). We put $V_i^{(2^\lambda),\text{ABS}}$ in a red block (e.g., $V_2^{(4),\text{ABS}}$) if $2i \in \mathcal{I}^{(2^{\lambda+1})}$. In this case, `decode_channel`(λ, i) calls `decode_swapped_channel`(λ, i). An arrow from $V_i^{(2^\lambda),\text{ABS}}$ to $V_{i_1}^{(2^{\lambda-1}),\text{ABS}}$ means that `decode_channel`(λ_1, i_1) is called in the execution of `decode_channel`(λ, i). For example, we call `decode_channel` with input parameters (2, 1), (2, 2) and (2, 3) in the execution of `decode_channel`(1, 1).

C. SCL decoder for ABS polar codes

In this subsection, we present the new SCL decoder for ABS polar codes. This decoder is based on the DB polar transform in Fig. 3 and the SDB polar transform in Fig. 5. Since we have the permutation matrices $\mathbf{P}_2^{\text{ABS}}, \mathbf{P}_4^{\text{ABS}}, \dots, \mathbf{P}_n^{\text{ABS}}$ in the ABS polar code construction, we need to replace the recursive relation (26) with

$$(X_1^{(2^\lambda)}, X_2^{(2^\lambda)}, \dots, X_n^{(2^\lambda)}) = (X_1^{(2^{\lambda+1})}, X_2^{(2^{\lambda+1})}, \dots, X_n^{(2^{\lambda+1})}) \cdot ((\mathbf{P}_{2^{\lambda+1}}^{\text{ABS}}(\mathbf{I}_{2^\lambda} \otimes \mathbf{G}_2^{\text{polar}})) \otimes \mathbf{I}_{2^{m-\lambda-1}}) \quad (43)$$

in order to define the intermediate vectors in ABS polar codes. We still use the notation in (27) and (31).

The data structures in this subsection are essentially the same as the ones in the previous subsection. There are only two minor differences:

(i) We change the range of the index s in (37) from $1 \leq s \leq 4L$ to $1 \leq s \leq 6L$.

(ii) In the integer array \mathbb{N}_B , each entry $\mathbb{N}_B[\lambda]$ takes value in $\{0, 1, 2, \dots, 6L\}$ instead of $\{0, 1, 2, \dots, 4L\}$.

For the SCL decoder presented in this subsection, the fields of the ℓ th list element are the same as the ones listed in (38).

The following functions are shared by the decoder in this subsection and the decoders in previous sections:

- (1) `allocate_prob` in Algorithm 4
- (2) `allocate_bit` in Algorithm 11
- (3) `decode` in Algorithm 12. This is the main function of the decoder.
- (4) `calculate_nabla_transform` in Algorithm 14
- (5) `calculate_nabla_transform` in Algorithm 15
- (6) `calculate_nabla_transform` in Algorithm 16
- (7) `decode_boundary_channel` in Algorithm 17

The following functions are solely used in this subsection. More precisely, either they appeared in previous subsections with the same name but with different implementations or they did not appear in previous subsections at all.

- (1) `decode_channel` in Algorithm 19. In Section V-B, we also have the function `decode_channel` in Algorithm 13, but the implementations in these two algorithms are different.

- (2) `decode_original_channel` in Algorithm 20. In Section V-B, we also have the function `decode_original_channel` in Algorithm 18, but the implementations in these two algorithms are different.
- (3) `decode_swapped_channel` in Algorithm 21. This function did not appear in previous subsections.
- (4) `calculate_∇_transform` in Algorithm 22. This function did not appear in previous subsections.
- (5) `calculate_◆_transform` in Algorithm 23. This function did not appear in previous subsections.
- (6) `calculate_▲_transform` in Algorithm 24. This function did not appear in previous subsections.

Although this subsection and the previous subsection share the same main function `decode` in Algorithm 12, the function `decode_channel` in Line 8 of Algorithm 12 has different implementations in these two subsections. More specifically, the function `decode_channel` in Algorithm 19 has one more branch than `decode_channel` in Algorithm 13. The additional branch decodes swapped adjacent bits.

Algorithm 20 and Algorithm 18 are the implementations of `decode_original_channel` for this subsection and the previous subsection, respectively. The difference between Algorithm 20 and Algorithm 18 is that we calculate the ∇ transform only when $2(i-1) \notin \mathcal{I}^{(2^{\lambda+1})}$ in Algorithm 20; see Lines 2–6. In contrast, we always calculate the ∇ transform in Algorithm 18; see Lines 2–5. The reason behind this difference is given in Lemma 2: When $2(i-1) \in \mathcal{I}^{(2^{\lambda+1})}$, we only have $V_{2i-1}^{(2^{\lambda+1}),\text{ABS}} = (V_{i-1}^{(2^\lambda),\text{ABS}})^\blacktriangle$, but $V_{2i-1}^{(2^{\lambda+1}),\text{ABS}} = (V_i^{(2^\lambda),\text{ABS}})^\nabla$ does not hold, so we do not calculate the ∇ transform in this case.

In Fig. 9, we use the ABS polar code defined in Fig. 7 as a concrete example to illustrate the recursive structure of the function `decode_channel` in Algorithm 19.

Lemma 7. *Suppose that $1 \leq \lambda \leq m$ and $1 \leq i \leq 2^\lambda - 1$. Before we call the function `decode_channel` in Algorithm 19 with input parameters (λ, i) , the pointer $\mathbb{P}[\ell, \lambda]$ satisfies that*

$$\mathbb{P}[\ell, \lambda][\beta, a, b] = V_i^{(2^\lambda)}(\hat{\mathbf{o}}_{i,\beta}^{(\ell,\lambda)} | a, b) \quad \text{for all } 1 \leq \ell \leq L_c, 1 \leq \beta \leq 2^{m-\lambda} \text{ and } a, b \in \{0, 1\}. \quad (44)$$

After the function `decode_channel`(λ, i) in Algorithm 19 returns, the pointer $\mathbb{R}[\ell, \lambda]$ satisfies that

$$\mathbb{R}[\ell, \lambda][\beta] = \hat{x}_{i,\beta}^{(\ell,\lambda)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda}. \quad (45)$$

Moreover, if $i = 2^\lambda - 1$, then the pointer $\mathbb{R}[\ell, \lambda]$ further satisfies that

$$\mathbb{R}[\ell, \lambda][\beta + 2^{m-\lambda}] = \hat{x}_{i+1,\beta}^{(\ell,\lambda)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda}. \quad (46)$$

Proof. The proof of (44) is the same as that of (39). Here we only prove (45)–(46) by induction. The proof of the base case $\lambda = m$ relies on the analysis of Algorithm 17, which was already done in the proof of Lemma 5. For the inductive step, we assume that (45)–(46) hold for $\lambda + 1$ and prove them for λ . This requires us to analyze Algorithm 20 for $2i \notin \mathcal{I}^{(2^{\lambda+1})}$ and analyze Algorithm 21 for $2i \in \mathcal{I}^{(2^{\lambda+1})}$. Algorithm 20 and Algorithm 18 are essentially the same. Since we have already analyzed Algorithm 18 in the proof of Lemma 5, we omit the analysis of Algorithm 20 here. We will focus on the analysis of Algorithm 21 for the rest of this proof.

By the induction hypothesis, after executing Lines 1–5 of Algorithm 21, we have

$$\mathbb{H}[\ell, \lambda + 1][\beta] = \hat{x}_{2i-1,\beta}^{(\ell,\lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

After executing Lines 7–10 and Lines 17,27, we have

$$\text{temp}[\beta] = \hat{x}_{2i,\beta}^{(\ell,\lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

After executing Lines 12–13, we have

$$\mathbb{R}[\ell, \lambda + 1][\beta] = \hat{x}_{2i+1,\beta}^{(\ell,\lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \quad (47)$$

In Line 1, we set $n_c = 2^\lambda$. We again divide the discussion into two cases. **Case (1)** $i \leq n_c - 2$: In this case, we only need to prove (45). Since $n_c = 2^\lambda$, we have $n/(2n_c) = 2^{m-\lambda-1}$. Therefore, Lines 21–22 of Algorithm 21 become

$$\begin{aligned} \mathbf{R}[\ell, \lambda][\beta] &= \hat{x}_{2i-1, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)}, & \mathbf{R}[\ell, \lambda][\beta + 2^{m-\lambda-1}] &= \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)} \\ & \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned} \quad (48)$$

Equations (43) and (27) together imply that if $2i \in \mathcal{I}^{(2^{\lambda+1})}$, then

$$X_{i, \beta}^{(\lambda)} = X_{2i-1, \beta}^{(\lambda+1)} + X_{2i+1, \beta}^{(\lambda+1)}, \quad X_{i, \beta+2^{m-\lambda-1}}^{(\lambda)} = X_{2i+1, \beta}^{(\lambda+1)} \quad \text{for all } 1 \leq \beta \leq 2^{m-\lambda-1}, \quad (49)$$

$$X_{i+1, \beta}^{(\lambda)} = X_{2i, \beta}^{(\lambda+1)} + X_{2i+2, \beta}^{(\lambda+1)}, \quad X_{i+1, \beta+2^{m-\lambda-1}}^{(\lambda)} = X_{2i+2, \beta}^{(\lambda+1)} \quad \text{for all } 1 \leq \beta \leq 2^{m-\lambda-1}. \quad (50)$$

(49) further implies that

$$\begin{aligned} \hat{x}_{i, \beta}^{(\ell, \lambda)} &= \hat{x}_{2i-1, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)}, & \hat{x}_{i, \beta+2^{m-\lambda-1}}^{(\ell, \lambda)} &= \hat{x}_{2i+1, \beta}^{(\ell, \lambda+1)} \\ & \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned}$$

Combining this with (48), we complete the proof of (45) for Case (1). **Case (2)** $i = n_c - 1$: In this case, we need to prove both (45) and (46). The proof of (45) is exactly the same as Case (1). To prove (46), we observe that if $i = n_c - 1$, then $2i + 1 = 2n_c - 1 = 2^{\lambda+1} - 1$. Then by the induction hypothesis, after executing Lines 12–13, we have not only (47) but also

$$\mathbf{R}[\ell, \lambda + 1][\beta + 2^{m-\lambda-1}] = \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)} \quad \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}.$$

Therefore, Lines 33–35 become

$$\begin{aligned} \mathbf{R}[\ell, \lambda][\beta + 2^{m-\lambda}] &= \hat{x}_{2i, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)}, & \mathbf{R}[\ell, \lambda][\beta + 2^{m-\lambda-1} + 2^{m-\lambda}] &= \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)} \\ & \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned} \quad (51)$$

Equation (50) implies that

$$\begin{aligned} \hat{x}_{i+1, \beta}^{(\ell, \lambda)} &= \hat{x}_{2i, \beta}^{(\ell, \lambda+1)} + \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)}, & \hat{x}_{i+1, \beta+2^{m-\lambda-1}}^{(\ell, \lambda)} &= \hat{x}_{2i+2, \beta}^{(\ell, \lambda+1)} \\ & \text{for all } 1 \leq \ell \leq L_c \text{ and } 1 \leq \beta \leq 2^{m-\lambda-1}. \end{aligned}$$

Combining this with (51), we complete the proof of (46). \square

The next lemma shows that the data structures \mathbf{D} and \mathbf{B} are large enough to store the transition probabilities and the decoding results of the intermediate vectors throughout the decoding procedure.

Lemma 8. *Throughout the whole decoding procedure, we have $N_{\mathbf{D}}[\lambda] \leq L$ and $N_{\mathbf{B}}[\lambda] \leq 6L$ for all $1 \leq \lambda \leq m$. The space complexity of the SCL decoder is $O(Ln)$.*

Proof. In TABLE II, we use the method in the proof of Lemma 6 to obtain the upper bound of $N_{\mathbf{B}}[\lambda + 1]$ at the starting point and the end of the function `decode_channel`($\lambda + 1, j$). The upper bounds in TABLE II immediately imply $N_{\mathbf{B}}[\lambda] \leq 6L$. The proof of $N_{\mathbf{D}}[\lambda] \leq L$ is the same as Lemma 4, and we do not repeat it.

The proof of the space complexity is the same as Lemma 4. \square

cases			start	end
$2i \in \mathcal{I}^{(2^{\lambda+1})}$	$1 \leq i \leq 2^\lambda - 1$	$j = 2i - 1$	0	L
		$j = 2i$	L	2L
	$1 \leq i < 2^\lambda - 1$	$j = 2i + 1$	2L	3L
$i = 2^\lambda - 1$	2L		4L	
$2(i - 1) \in \mathcal{I}^{(2^{\lambda+1})}$	$1 \leq i \leq 2^\lambda - 1$	$j = 2i$	3L	4L
	$i = 2^\lambda - 1$	$j = 2i + 1$	4L	6L
$2(i - 1) \notin \mathcal{I}^{(2^{\lambda+1})}, 2i \notin \mathcal{I}^{(2^{\lambda+1})}$	$1 \leq i \leq 2^\lambda - 1$	$j = 2i - 1$	0	L
		$j = 2i$	L	2L
	$i = 2^\lambda - 1$	$j = 2i + 1$	2L	4L

TABLE II: The upper bound of $\mathbb{N}_{\mathbb{B}}[\lambda + 1]$ at the starting point and the end of the function $\text{decode_channel}(\lambda + 1, j)$

Proposition 4. *The decoding time complexity of ABS polar codes is $O(Ln \log(n))$.*

Algorithm 17: decode_boundary_channel(i)

Input: index i in the last layer ($\lambda = m$)

```

1 flag  $\leftarrow$  1
2 if  $i \leq n - 2$  then
3   if  $i \in \mathcal{A}$  then
4     for  $\ell \in \{1, 2, \dots, L_c\}$ ,  $a \in \{0, 1\}$  do
5       prob  $\leftarrow \frac{1}{2} \sum_{b \in \{0, 1\}} P[\ell, m][1, a, b]$ 
6       PriQue.push( $\ell, a, \text{"?"}, \text{prob}$ )
7   else
8     flag  $\leftarrow$  0
9     for  $\ell \in \{1, 2, \dots, L_c\}$  do
10       $R[\ell, m] \leftarrow \text{allocate\_bit}(m, 1)$ 
11       $R[\ell, m][1] \leftarrow \text{frozen value of } U_i$ 
12 else
13   if  $n - 1, n \notin \mathcal{A}$  then
14     flag  $\leftarrow$  0
15     for  $\ell \in \{1, 2, \dots, L_c\}$  do
16       $R[\ell, m] \leftarrow \text{allocate\_bit}(m, 2)$ 
17       $(R[\ell, m][1], R[\ell, m][2]) \leftarrow (U_{n-1}, U_n)$ 
18   else if  $n - 1 \in \mathcal{A}$ ,  $n \notin \mathcal{A}$  then
19     for  $\ell \in \{1, 2, \dots, L_c\}$ ,  $a \in \{0, 1\}$  do
20       PriQue.push( $\ell, a, U_n, P[\ell, m][1, a, U_n]$ )
21   else if  $n - 1 \notin \mathcal{A}$ ,  $n \in \mathcal{A}$  then
22     for  $\ell \in \{1, 2, \dots, L_c\}$ ,  $b \in \{0, 1\}$  do
23       PriQue.push( $\ell, U_{n-1}, b, P[\ell, m][1, U_{n-1}, b]$ )
24   else
25     for  $\ell \in \{1, 2, \dots, L_c\}$ ,  $a, b \in \{0, 1\}$  do
26       PriQue.push( $\ell, a, b, P[\ell, m][1, a, b]$ )
27
28
29 if flag = 1 then
30    $L_c \leftarrow \min\{L, \text{PriQue.size}()\}$ 
31   for  $\ell \in \{1, 2, \dots, L_c\}$  do
32      $(\ell', a, b, \text{score}[\ell]) \leftarrow \text{PriQue.pop}()$ 
33     for  $\lambda \in \{1, 2, \dots, m - 1\}$  do
34        $\bar{P}[\ell, \lambda] \leftarrow P[\ell', \lambda]$ 
35        $\bar{R}[\ell, \lambda] \leftarrow R[\ell', \lambda]$ 
36        $\bar{H}[\ell, \lambda] \leftarrow H[\ell', \lambda]$ 
37     if  $i < n - 1$  then
38        $\bar{R}[\ell, m] \leftarrow \text{allocate\_bit}(m, 1)$ 
39        $\bar{R}[\ell, m][1] \leftarrow a$ 
40     else
41        $\bar{R}[\ell, m] \leftarrow \text{allocate\_bit}(m, 2)$ 
42        $(\bar{R}[\ell, m][1], \bar{R}[\ell, m][2]) \leftarrow (a, b)$ 
43   PriQue.clear()
44   swap( $\bar{P}, P$ ), swap( $\bar{R}, R$ ), swap( $\bar{H}, H$ )
45 return
```

Algorithm 18: decode_original_channel(λ, i)

Input: $\lambda \in \{1, 2, \dots, m\}$ and index i satisfying $1 \leq i \leq 2^\lambda - 1$

- 1 $n_c \leftarrow 2^\lambda$ $\triangleright V_{2i-1}^{(2n_c)} = (V_i^{(n_c)})^\nabla$
- 2 calculate_∇_transform($\lambda + 1$)
- 3 decode_channel($\lambda + 1, 2i - 1$)
- 4 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 5 $\lfloor H[\ell, \lambda + 1] \leftarrow R[\ell, \lambda + 1]$
- 6 $\triangleright V_{2i}^{(2n_c)} = (V_i^{(n_c)})^\diamond$
- 7 calculate_◇_transform($\lambda + 1$)
- 8 decode_channel($\lambda + 1, 2i$)
- 9 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 10 $\lfloor R[\ell, \lambda] \leftarrow R[\ell, \lambda + 1]$
- 11 **if** $i \leq n_c - 2$ **then**
- 12 \triangleright Only decode one bit $X_{i,\beta}^{(\lambda)}$ for each β
- 13 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 14 temp $\leftarrow R[\ell, \lambda]$
- 15 $R[\ell, \lambda] \leftarrow \text{allocate_bit}(\lambda, 1)$
- 16 **for** $\beta \in \{1, 2, \dots, n/(2n_c)\}$ **do**
- 17 $\beta' \leftarrow \beta + n/(2n_c)$
- 18 $R[\ell, \lambda][\beta] \leftarrow H[\ell, \lambda + 1][\beta] + \text{temp}[\beta]$
- 19 $R[\ell, \lambda][\beta'] \leftarrow \text{temp}[\beta]$
- 20 **else**
- 21 \triangleright Decode two bits $X_{n_c-1,\beta}^{(\lambda)}, X_{n_c,\beta}^{(\lambda)}$ for each β
- 22 $\triangleright V_{2i+1}^{(2n_c)} = (V_i^{(n_c)})^\Delta$
- 23 calculate_Δ_transform($\lambda + 1$)
- 24 decode_channel($\lambda + 1, 2i + 1$)
- 25 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 26 temp $\leftarrow R[\ell, \lambda]$
- 27 $R[\ell, \lambda] \leftarrow \text{allocate_bit}(\lambda, 2)$
- 28 **for** $\beta \in \{1, 2, \dots, n/(2n_c)\}$ **do**
- 29 $\beta' \leftarrow \beta + n/(2n_c)$
- 30 $R[\ell, \lambda][\beta] \leftarrow H[\ell, \lambda + 1][\beta] + \text{temp}[\beta]$
- 31 $R[\ell, \lambda][\beta'] \leftarrow \text{temp}[\beta]$
- 32 $R[\ell, \lambda][\beta + n/(n_c)] \leftarrow$
- 33 $R[\ell, \lambda + 1][\beta] + R[\ell, \lambda + 1][\beta']$
- 34 $R[\ell, \lambda][\beta' + n/(n_c)] \leftarrow R[\ell, \lambda + 1][\beta']$
- 35 $N_B[\lambda + 1] \leftarrow 0$
- 36 **return**

Algorithm 19: `decode_channel(λ, i)`

Input: layer $\lambda \in \{1, 2, \dots, m\}$ and index $i \in \{1, 2, \dots, 2^\lambda - 1\}$

```

1 if  $\lambda = m$  then
2   | decode_boundary_channel( $i$ ) ▷ Algorithm 17
3 else if  $2i \notin \mathcal{I}^{(2^{\lambda+1})}$  then
4   | decode_original_channel( $\lambda, i$ ) ▷ Algorithm 20
5 else if  $2i \in \mathcal{I}^{(2^{\lambda+1})}$  then
6   | decode_swapped_channel( $\lambda, i$ ) ▷ Algorithm 21
7  $N_D[\lambda] \leftarrow 0$ 
8 return

```

Algorithm 20: decode_original_channel(λ, i)

Input: $\lambda \in \{1, 2, \dots, m\}$ and index i satisfying $1 \leq i \leq 2^\lambda - 1$ and $2i \notin \mathcal{I}^{(2^{\lambda+1})}$

- 1 $n_c \leftarrow 2^\lambda$
- 2 **if** $2(i-1) \notin \mathcal{I}^{(2^{\lambda+1})}$ **then** $\triangleright V_{2i-1}^{(2n_c)} = (V_i^{(n_c)})^\nabla$
- 3 calculate_∇_transform($\lambda+1$)
- 4 decode_channel($\lambda+1, 2i-1$)
- 5 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 6 $H[\ell, \lambda+1] \leftarrow R[\ell, \lambda+1]$
- 7 $\triangleright V_{2i}^{(2n_c)} = (V_i^{(n_c)})^\diamond$
- 8 calculate_◇_transform($\lambda+1$)
- 9 decode_channel($\lambda+1, 2i$)
- 10 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 11 $R[\ell, \lambda] \leftarrow R[\ell, \lambda+1]$
- 12 **if** $i \leq n_c - 2$ **then**
- 13 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do** \triangleright Only decode one bit $X_{i,\beta}^{(\lambda)}$ for each β
- 14 temp $\leftarrow R[\ell, \lambda]$
- 15 $R[\ell, \lambda] \leftarrow \text{allocate_bit}(\lambda, 1)$
- 16 **for** $\beta \in \{1, 2, \dots, n/(2n_c)\}$ **do**
- 17 $\beta' \leftarrow \beta + n/(2n_c)$
- 18 $R[\ell, \lambda][\beta] \leftarrow H[\ell, \lambda+1][\beta] + \text{temp}[\beta]$
- 19 $R[\ell, \lambda][\beta'] \leftarrow \text{temp}[\beta]$
- 20
- 21 **else** \triangleright Decode two bits $X_{n_c-1,\beta}^{(\lambda)}, X_{n_c,\beta}^{(\lambda)}$ for each β
- 22 $\triangleright V_{2i+1}^{(2n_c)} = (V_i^{(n_c)})^\Delta$
- 23 calculate_Δ_transform($\lambda+1$)
- 24 decode_channel($\lambda+1, 2i+1$)
- 25 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 26 temp $\leftarrow R[\ell, \lambda]$
- 27 $R[\ell, \lambda] \leftarrow \text{allocate_bit}(\lambda, 2)$
- 28 **for** $\beta \in \{1, 2, \dots, n/(2n_c)\}$ **do**
- 29 $\beta' \leftarrow \beta + n/(2n_c)$
- 30 $R[\ell, \lambda][\beta] \leftarrow H[\ell, \lambda+1][\beta] + \text{temp}[\beta]$
- 31 $R[\ell, \lambda][\beta'] \leftarrow \text{temp}[\beta]$
- 32 $R[\ell, \lambda][\beta + n/(n_c)] \leftarrow$
- 33 $R[\ell, \lambda+1][\beta] + R[\ell, \lambda+1][\beta']$
- 34 $R[\ell, \lambda][\beta' + n/(n_c)] \leftarrow R[\ell, \lambda+1][\beta']$
- 35
- 36 $N_B[\lambda+1] \leftarrow 0$
- 37 **return**

Algorithm 21: decode_swapped_channel(λ, i)

Input: $\lambda \in \{1, 2, \dots, m\}$ and index i satisfying $1 \leq i \leq 2^{m-\lambda} - 1$ and $2i \in \mathcal{I}^{(2^{\lambda+1})}$.

- 1 $n_c \leftarrow 2^\lambda$ $\triangleright V_{2i-1}^{(2n_c)} = (V_i^{(n_c)})^\nabla$
- 2 calculate_ \blacktriangledown _transform($\lambda + 1$)
- 3 decode_channel($\lambda + 1, 2i - 1$)
- 4 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 5 $\lfloor H[\ell, \lambda + 1] \leftarrow R[\ell, \lambda + 1]$

- 6 $\triangleright V_{2i}^{(2n_c)} = (V_i^{(n_c)})^\blacklozenge$
- 7 calculate_ \blacklozenge _transform($\lambda + 1$)
- 8 decode_channel($\lambda + 1, 2i$)
- 9 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 10 $\lfloor R[\ell, \lambda] \leftarrow R[\ell, \lambda + 1]$

- 11 $\triangleright V_{2i+1}^{(2n_c)} = (V_i^{(n_c)})^\blacktriangle$
- 12 calculate_ \blacktriangle _transform($\lambda + 1$)
- 13 decode_channel($\lambda + 1, 2i + 1$)
- 14 **if** $i \leq n_c - 2$ **then**
- 15 \triangleright Only decode one bit $X_{i,\beta}^{(\lambda)}$ for each β
- 16 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 17 temp $\leftarrow R[\ell, \lambda]$
- 18 $R[\ell, \lambda] \leftarrow \text{allocate_bit}(\lambda, 1)$
- 19 **for** $\beta \in \{1, 2, \dots, n/(2n_c)\}$ **do**
- 20 $\beta' \leftarrow \beta + n/(2n_c)$
- 21 $R[\ell, \lambda][\beta] \leftarrow H[\ell, \lambda + 1][\beta] + R[\ell, \lambda + 1][\beta]$
- 22 $R[\ell, \lambda][\beta'] \leftarrow R[\ell, \lambda + 1][\beta]$
- 23 $H[\ell, \lambda + 1] \leftarrow \text{temp}$
- 24 **else**
- 25 \triangleright Decode two bits $X_{n_c-1,\beta}^{(\lambda)}, X_{n_c,\beta}^{(\lambda)}$ for each β
- 26 **for** $\ell \in \{1, 2, \dots, L_c\}$ **do**
- 27 temp $\leftarrow R[\ell, \lambda]$
- 28 $R[\ell, \lambda] \leftarrow \text{allocate_bit}(\lambda, 2)$
- 29 **for** $\beta \in \{1, 2, \dots, n/(2n_c)\}$ **do**
- 30 $\beta' \leftarrow \beta + n/(2n_c)$
- 31 $R[\ell, \lambda][\beta] \leftarrow H[\ell, \lambda + 1][\beta] + R[\ell, \lambda + 1][\beta]$
- 32 $R[\ell, \lambda][\beta'] \leftarrow R[\ell, \lambda + 1][\beta]$
- 33 $R[\ell, \lambda][\beta + n/(n_c)] \leftarrow$
- 34 temp $[\beta] + R[\ell, \lambda + 1][\beta']$
- 35 $R[\ell, \lambda][\beta' + n/(n_c)] \leftarrow R[\ell, \lambda + 1][\beta']$
- 36 $N_B[\lambda + 1] \leftarrow 0$
- 37 **return**

Algorithm 22: calculate_▾_transform(λ)

Input: layer $2 \leq \lambda \leq m$ **Output:** Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

```

1  $\bar{n}_c \leftarrow 2^{m-\lambda}$ 
2 for  $\ell \in \{1, 2, \dots, L_c\}$  do
3    $P[\ell, \lambda] \leftarrow \text{allocate\_prob}(\lambda)$ 
4   for  $\beta \in \{1, 2, \dots, \bar{n}_c\}, r_1, r_2 \in \{0, 1\}$  do
5      $\beta' \leftarrow \beta + \bar{n}_c$ 
6      $P[\ell, \lambda][\beta, r_1, r_2] \leftarrow \frac{1}{4} \sum_{r_3, r_4 \in \{0, 1\}} P[\ell, \lambda - 1][\beta, r_1 + r_3, r_2 + r_4] P[\ell, \lambda - 1][\beta', r_3, r_4]$ 
7 return

```

Algorithm 23: calculate_◆_transform(λ)

Input: layer $2 \leq \lambda \leq m$ **Output:** Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

```

1  $\bar{n}_c \leftarrow 2^{m-\lambda}$ 
2 for  $\ell \in \{1, 2, \dots, L_c\}$  do
3    $P[\ell, \lambda] \leftarrow \text{allocate\_prob}(\lambda)$ 
4   for  $\beta \in \{1, 2, \dots, \bar{n}_c\}, r_2, r_3 \in \{0, 1\}$  do
5      $r_1 \leftarrow \mathbf{H}[\ell, \lambda][\beta]$ 
6      $\beta' \leftarrow \beta + \bar{n}_c$ 
7      $P[\ell, \lambda][\beta, r_2, r_3] \leftarrow \frac{1}{4} \sum_{r_4 \in \{0, 1\}} P[\ell, \lambda - 1][\beta, r_1 + r_3, r_2 + r_4] P[\ell, \lambda - 1][\beta', r_3, r_4]$ 
8 return

```

Algorithm 24: calculate_▲_transform(λ)

Input: layer $2 \leq \lambda \leq m$ **Output:** Update the entries pointed by $P[\ell, \lambda]$, $1 \leq \ell \leq L_c$

```

1  $\bar{n}_c \leftarrow 2^{m-\lambda}$ 
2 for  $\ell \in \{1, 2, \dots, L_c\}$  do
3    $P[\ell, \lambda] \leftarrow \text{allocate\_prob}(\lambda)$ 
4   for  $\beta \in \{1, 2, \dots, \bar{n}_c\}, r_3, r_4 \in \{0, 1\}$  do
5      $r_1 \leftarrow \mathbf{H}[\ell, \lambda][\beta], r_2 \leftarrow \mathbf{R}[\ell, \lambda - 1][\beta]$ 
6      $\beta' \leftarrow \beta + \bar{n}_c$ 
7      $P[\ell, \lambda][\beta, r_3, r_4] \leftarrow$ 
8      $\frac{1}{4} P[\ell, \lambda - 1][\beta, r_1 + r_3, r_2 + r_4] P[\ell, \lambda - 1][\beta', r_3, r_4]$ 
9 return

```

VI. SIMULATION RESULTS

A. Scaling exponent over binary erasure channels

In this subsection, we empirically calculate the scaling exponents of ABS polar codes and standard polar codes over a BEC with erasure probability 0.5.

When the original channel W is a general BMS channel, we can only obtain an approximation of the transition probabilities of the adjacent-bits-channels through quantization, as discussed in Section III-E. However, when the original channel W is a BEC, we are able to calculate the exact parameters of the adjacent-bits-channels. To that end, we introduce a class of channels called double-bits-erasure-channels (DBEC). The input alphabet of a DBEC is $\{0, 1\}^2$, and the output alphabet is $\{0, 1, ?\}^3$. For a given input $(u_1, u_2) \in \{0, 1\}^2$, the output of the DBEC can only take the following five values

- $(u_1, u_1 + u_2, u_2)$ with probability p ,
- $(u_1, ?, ?)$ with probability q ,
- $(?, u_1 + u_2, ?)$ with probability r ,
- $(?, ?, u_2)$ with probability s ,
- $(?, ?, ?)$ with probability t .

p is the probability of preserving all information in the inputs. q, r, s are the probabilities of preserving one bit of information in the inputs. t is the probability of erasing all the information. Such a DBEC is denoted as $\text{DBEC}(p, q, r, s, t)$, where the parameters satisfy $p + q + r + s + t = 1$. Note that DBEC has been studied in the literature under other names. For example, the authors of [29] call it tetrahedral erasure channel.

One can show that if the original channel W is a BEC, then all the adjacent-bits-channels in the ABS polar code construction are DBEC. More precisely, using (22), we can show that if W is a BEC with erasure probability ϵ , then

$$V_1^{(2), \text{ABS}} = \text{DBEC}((1 - \epsilon)^2, 0, (1 - \epsilon)\epsilon, (1 - \epsilon)\epsilon, \epsilon^2).$$

Moreover, if an adjacent-bits-channel $V = \text{DBEC}(p, q, r, s, t)$, then

$$\begin{aligned} V^\nabla &= \text{DBEC}((p + q)^2, 0, (p + q)(r + s + t), (r + s + t)(p + q), (r + s + t)^2), \\ V^\diamond &= \text{DBEC}(p^2 + 2rp + 2sp, 2q - q^2 + 2pt, 2rs, r^2 + s^2, 2t(r + s) + t^2), \\ V^\Delta &= \text{DBEC}((p + r + s)^2, 0, (p + r + s)(q + t), (q + t)(p + r + s), (q + t)^2), \\ V^\nabla &= \text{DBEC}(p^2, q^2 + 2pq, r^2 + 2pr, s^2 + 2ps, 2t - t^2 + 2qr + 2qs + 2rs), \\ V^\blacklozenge &= \text{DBEC}(p^2 + 2pr + 2ps, r^2 + s^2, 2rs, 2q - q^2 + 2pt, t^2 + 2rt + 2rs), \\ V^\blacktriangle &= \text{DBEC}(2p - p^2 + 2qr + 2qs + 2rs, q^2 + 2tq, r^2 + 2rt, s^2 + 2st, t^2). \end{aligned}$$

Combining this with Lemma 2, we can explicitly calculate the parameters of all the adjacent-bits-channels in the ABS polar code construction when the original channel W is a BEC. After that, we use (19) to calculate the erasure probabilities of each bit-channel: If $V_i^{(n), \text{ABS}} = \text{DBEC}(p, q, r, s, t)$, then $W_i^{(n), \text{ABS}}$ is an erasure channel with erasure probability $r + s + t$, and $W_{i+1}^{(n), \text{ABS}}$ is an erasure channel with erasure probability $q + t$.

Let W be a BEC with erasure probability 0.5. For $n \in \{2^6, 2^7, \dots, 2^{20}\}$, we define

$$\begin{aligned} f_{\text{polar}}(n) &:= \frac{1}{n} |\{i : 1 \leq i \leq n, \quad 0.01 \leq I(W_i^{(n)}) \leq 0.99\}|, \\ f_{\text{ABS}}(n) &:= \frac{1}{n} |\{i : 1 \leq i \leq n, \quad 0.01 \leq I(W_i^{(n), \text{ABS}}) \leq 0.99\}|. \end{aligned}$$

By definition, $f_{\text{polar}}(n)$ is the fraction of “unpolarized” bit-channels in the length- n standard polar code constructed for the BEC W , and $f_{\text{ABS}}(n)$ is the fraction of “unpolarized” bit-channels in the length- n ABS polar code constructed for the BEC W . A bit-channel is said to be unpolarized if its capacity is

n	$f_{\text{polar}}(n)$	$f_{\text{ABS}}(n)$
64	0.53125000	0.50000000
128	0.43750000	0.42187500
256	0.37500000	0.34375000
512	0.30078125	0.27343750
1024	0.25390625	0.22070312
2048	0.20605469	0.18164062
4096	0.17041016	0.15136719
8192	0.14208984	0.12329102
16384	0.11755371	0.09936523
32768	0.09674072	0.08087158
65536	0.07995605	0.06542969
131072	0.06613159	0.05333710
262144	0.05499268	0.04324722
524288	0.04529572	0.03502846
1048576	0.03742218	0.02853012

TABLE III: The fractions of “unpolarized” bit-channels in standard polar codes and ABS polar codes constructed for a BEC with erasure probability $\epsilon = 0.5$.

(n, k)	(256, 77)	(256, 128)	(256, 179)	(512, 154)	(512, 256)	(512, 358)
ST, $L = 32$	0.963ms	1.41ms	1.73ms	1.94ms	2.80ms	3.54ms
ABS, $L = 20$	0.816ms	1.24ms	1.47ms	1.86ms	2.66ms	3.10ms
ABS, $L = 32$	1.29ms	1.99ms	2.37ms	2.93ms	4.36ms	5.13ms
(n, k)	(1024, 307)	(1024, 512)	(1024, 717)	(2048, 614)	(2048, 1024)	(2048, 1434)
ST, $L = 32$	4.21ms	5.75ms	7.15ms	9.05ms	11.7ms	14.6ms
ABS, $L = 20$	4.32ms	5.90ms	6.67ms	10.6ms	12.6ms	14.0ms
ABS, $L = 32$	6.63ms	9.41ms	10.8ms	16.7ms	20.1ms	23.2ms

TABLE IV: Comparison of the decoding time over the binary-input AWGN channel with $E_b/N_0 = 2$ dB. The row starting with (n, k) lists the code length and code dimension we have tested. The row starting with “ST, $L = 32$ ” lists the decoding time of the CRC-aided SCL decoder for standard polar codes with list size 32. The row starting with “ABS, $L = 20$ ” lists the decoding time of the CRC-aided SCL decoder for ABS polar codes with list size 20. The row starting with “ABS, $L = 32$ ” lists the decoding time of the CRC-aided SCL decoder for ABS polar codes with list size 32. The time unit “ms” is 10^{-3} s.

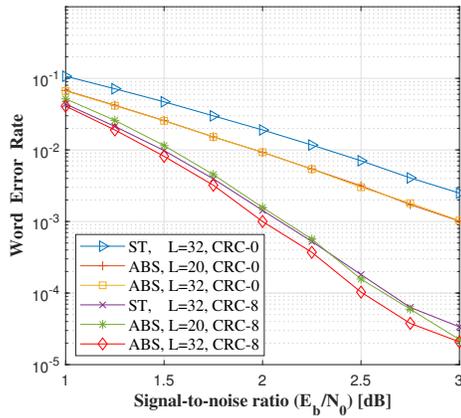
between 0.01 and 0.99. The values of $f_{\text{polar}}(n)$ and $f_{\text{ABS}}(n)$ for $n \in \{2^6, 2^7, \dots, 2^{20}\}$ are listed in TABLE III.

In order to estimate the scaling exponents, we approximate $f_{\text{polar}}(n)$ as $f_{\text{polar}}(n) \approx c_1 n^{-\gamma_1}$, and we approximate $f_{\text{ABS}}(n)$ as $f_{\text{ABS}}(n) \approx c_2 n^{-\gamma_2}$. By taking the logarithm on both sides of the equation and running linear regression, we obtain that $f_{\text{polar}}(n) \approx 1.67n^{-0.274}$ and $f_{\text{ABS}}(n) \approx 1.76n^{-0.297}$. Therefore, the scaling exponent for standard polar codes is $\mu_{\text{polar}} \approx 1/0.274 = 3.65$, and the scaling exponent for ABS polar codes is $\mu_{\text{ABS}} \approx 1/0.297 = 3.37$.

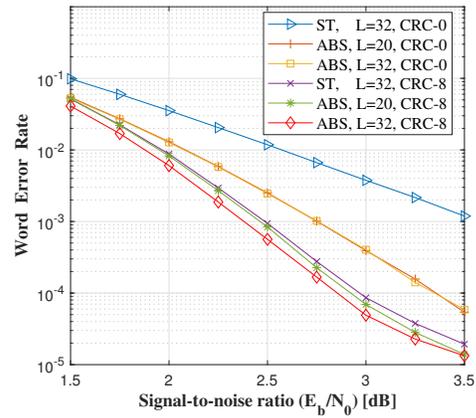
The above empirical calculations of scaling exponents confirm that the polarization of ABS polar codes is indeed faster than standard polar codes. An interesting problem for future research is to obtain provable and tight upper bounds on the scaling exponent of ABS polar codes. Another related question is to analyze the code distance of ABS polar codes and compare it with standard polar codes.

B. Simulation results over binary-input AWGN channels

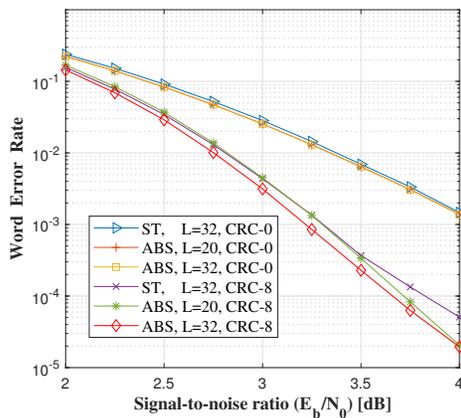
We conduct extensive simulations to compare the performance of the ABS polar codes and the standard polar codes over the binary-input AWGN channel with various choices of parameters. We have tested the performance for 4 different choices of code length 256, 512, 1024, 2048. For each choice of code length, we test 3 different code rates 0.3, 0.5 and 0.7. The comparison of decoding error probability is given in Fig. 10 and Fig. 11. Specifically, Fig. 10 contains the plots for code length 256 and 512; Fig. 11 contains the plots for code length 1024 and 2048. The comparison of decoding time is given in Table IV.



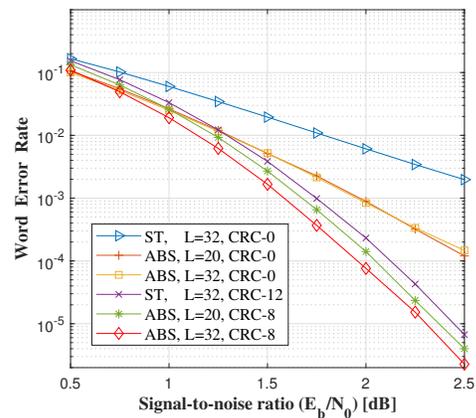
(a) length 256, dimension 77



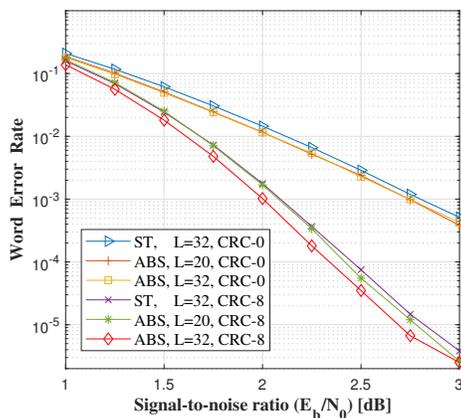
(b) length 256, dimension 128



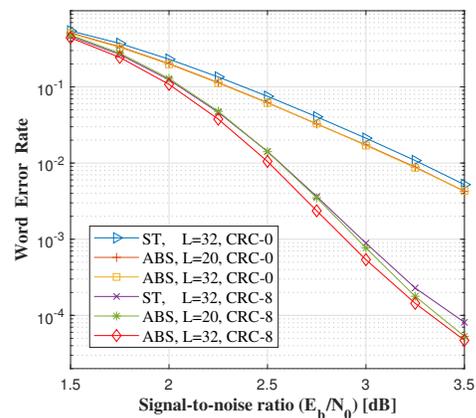
(c) length 256, dimension 179



(d) length 512, dimension 154

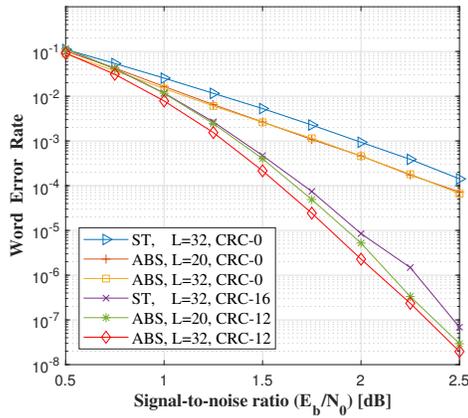


(e) length 512, dimension 256

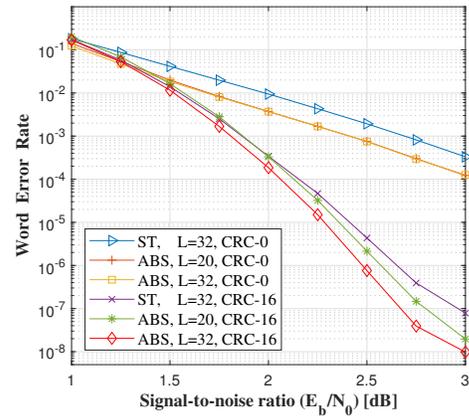


(f) length 512, dimension 358

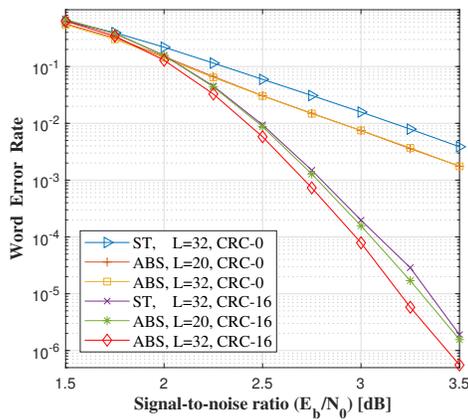
Fig. 10: Comparison between ABS polar codes and standard polar codes over the binary-input AWGN channel. The legend “ABS” refers to ABS polar codes, and “ST” refers to standard polar codes. “CRC-0” means that we do not use CRC. The nonzero CRC length is chosen from the set $\{4, 8, 12, 16, 20, 24\}$ to minimize the decoding error probability. The parameter L is the list size. For standard polar codes, we always choose $L = 32$. For ABS polar codes, we test two different list sizes $L = 20$ and $L = 32$.



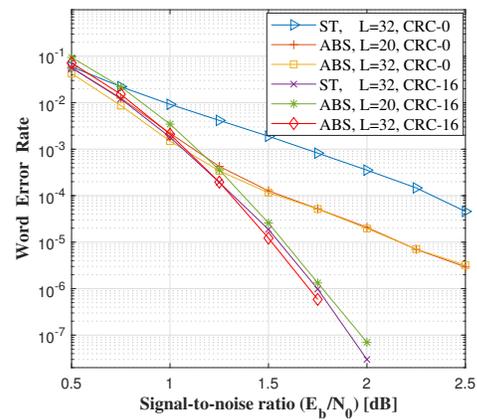
(a) length 1024, dimension 307



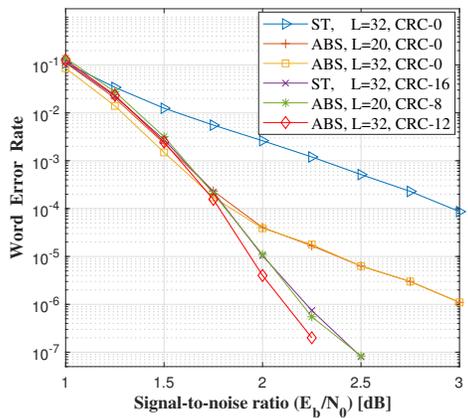
(b) length 1024, dimension 512



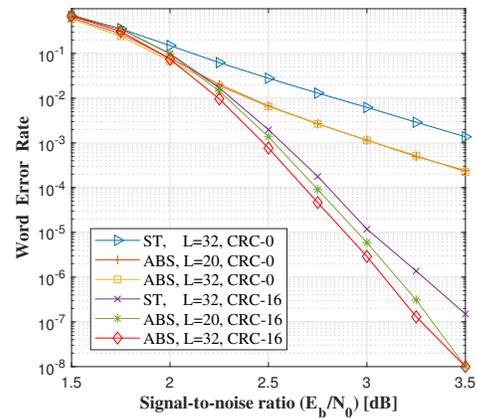
(c) length 1024, dimension 717



(d) length 2048, dimension 614



(e) length 2048, dimension 1024



(f) length 2048, dimension 1434

Fig. 11: Comparison between ABS polar codes and standard polar codes over the binary-input AWGN channel. The legend “ABS” refers to ABS polar codes, and “ST” refers to standard polar codes. “CRC-0” means that we do not use CRC. The nonzero CRC length is chosen from the set $\{4, 8, 12, 16, 20, 24\}$ to minimize the decoding error probability. The parameter L is the list size. For standard polar codes, we always choose $L = 32$. For ABS polar codes, we test two different list sizes $L = 20$ and $L = 32$.

In Fig. 10 and Fig. 11, for each choice of code length and code dimension, we compare the decoding error probability of the following 6 decoders: (1) SCL decoder for standard polar codes with list size 32 and no CRC; (2) SCL decoder for ABS polar codes with list size 20 and no CRC; (3) SCL decoder for ABS polar codes with list size 32 and no CRC; (4) SCL decoder for standard polar codes with list size 32 and optimal CRC length; (5) SCL decoder for ABS polar codes with list size 20 and optimal CRC length; (6) SCL decoder for ABS polar codes with list size 32 and optimal CRC length. The optimal CRC length is chosen from the set $\{4, 8, 12, 16, 20, 24\}$ to minimize the decoding error probability. For standard polar codes, we use the classic SCL decoder presented in Section V-A, **not** the new SCL decoder presented in Section V-B. For ABS polar codes, we use the SCL decoder presented in Section V-C.

Note that in a previous arXiv version and the ISIT version [1] of this paper, we used a different choice of CRC length. More specifically, for cases (4)–(6) in the above paragraph, we used CRC length 8 for all choices of code length and code dimension in the previous versions. In contrast, we use the optimal CRC length in this version, and the optimal CRC length varies with the code length and the code dimension.

From Fig. 10 and Fig. 11 we can see that the performance of ABS polar codes is consistently better than standard polar codes if we set the list size to be 32 for the CRC-aided SCL decoders of both codes. More specifically, for all 12 choices of (n, k) , the improvement of ABS polar codes over standard polar codes ranges from 0.15 dB to 0.3 dB. Even if we reduce the list size of ABS polar codes to be 20 and maintain the list size of standard polar codes to be 32, ABS polar codes still demonstrate better performance for most choices of parameters, and the improvement over standard polar codes is up to 0.15 dB in this case. Next let us compare the performance of ABS polar codes and standard polar codes when neither of them uses CRC. When there is no CRC, the performance of ABS polar codes with list size 20 is more or less the same as that of ABS polar codes with list size 32. Again, ABS polar codes consistently outperform standard polar codes for all 12 choices of (n, k) . This time the improvement over standard polar codes is up to 1.1 dB.

In Table IV, we only compare the decoding time of the SCL decoders with CRC length 8. From Table IV, we can see that the decoding time of the SCL decoder for ABS polar codes with list size 20 is more or less the same as the decoding time of the SCL decoder for standard polar codes with list size 32. More precisely, for 8 out of 12 choices of (n, k) , the SCL decoder for ABS polar codes with list size 20 runs faster. For the other 4 choices of (n, k) , the SCL decoder for standard polar codes with list size 32 runs faster. If we set the list size to be 32 for both the standard polar codes and the ABS polar codes, then Table IV tells us that the decoding time of ABS polar codes is longer than that of standard polar codes by roughly 60%.

In conclusion, when we use list size 32 for the CRC-aided SCL decoders of both codes, ABS polar codes consistently outperform standard polar codes by 0.15 dB—0.3 dB, but the decoding time of ABS polar decoder is longer than that of standard polar codes by roughly 60%. If we use list size 20 for ABS polar codes and maintain the list size to be 32 for standard polar codes, then the decoding time is more or less the same for these two codes, and ABS polar codes still outperform standard polar codes for most choices of parameters. In this case, the improvement over standard polar codes is up to 0.15 dB.

As a final remark, the implementations of all the algorithms in this paper are available at the website <https://github.com/PlumJelly/ABS-Polar>

ACKNOWLEDGEMENT

In the implementation of our decoding algorithm, we have learned a lot from the GitHub project <https://github.com/kshabunov/ecclab> maintained by Kirill Shabunov. Shabunov’s GitHub project mainly presents the implementation of the Reed-Muller decoder proposed in [12]. Due to the similarity between (ABS) polar codes and Reed-Muller codes, some of the accelerating techniques for Reed-Muller decoders can also be used to speed up (ABS) polar decoders.

APPENDIX A
THE PROOF OF LEMMA 1

Let (U_1, \dots, U_n) , (X_1, \dots, X_n) and (Y_1, \dots, Y_n) be the random vectors defined in Fig. 2. Define a new vector $(\tilde{U}_1, \dots, \tilde{U}_n)$ as follows:

$$\tilde{U}_{2i-1} = U_{2i-1} + U_{2i} \text{ and } \tilde{U}_{2i} = U_{2i} \text{ for all } 1 \leq i \leq n/2.$$

Since $\mathbf{G}_n^{\text{polar}} = \mathbf{G}_{n/2}^{\text{polar}} \otimes \mathbf{G}_2^{\text{polar}}$, we have

$$\begin{aligned} (X_1, X_3, X_5, \dots, X_{n-1}) &= (\tilde{U}_1, \tilde{U}_3, \tilde{U}_5, \dots, \tilde{U}_{n-1}) \mathbf{G}_{n/2}^{\text{polar}}, \\ (X_2, X_4, X_6, \dots, X_n) &= (\tilde{U}_2, \tilde{U}_4, \tilde{U}_6, \dots, \tilde{U}_n) \mathbf{G}_{n/2}^{\text{polar}}. \end{aligned}$$

Therefore, the mapping from $\tilde{U}_{2i-1}, \tilde{U}_{2i+1}$ to $\tilde{U}_1, \tilde{U}_3, \dots, \tilde{U}_{2i-3}, Y_1, Y_3, \dots, Y_{n-1}$ is $V_i^{(n/2)}$, and the channel mapping from $\tilde{U}_{2i}, \tilde{U}_{2i+2}$ to $\tilde{U}_2, \tilde{U}_4, \dots, \tilde{U}_{2i-2}, Y_2, Y_4, \dots, Y_n$ is also $V_i^{(n/2)}$. Moreover, the two random vectors $(\tilde{U}_1, \tilde{U}_3, \dots, \tilde{U}_{n-1}, Y_1, Y_3, \dots, Y_{n-1})$ and $(\tilde{U}_2, \tilde{U}_4, \dots, \tilde{U}_n, Y_2, Y_4, \dots, Y_n)$ are independent. As a consequence,

$$\begin{aligned} & V_{2i-1}^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i-2} | u_{2i-1}, u_{2i}) \\ &= \mathbb{P}_{Y_1, Y_2, \dots, Y_n, U_1, U_2, \dots, U_{2i-2} | U_{2i-1}, U_{2i}}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i-2} | u_{2i-1}, u_{2i}) \\ &= \frac{1}{4} \sum_{u_{2i+1}, u_{2i+2} \in \{0,1\}} \mathbb{P}_{Y_1, Y_2, \dots, Y_n, U_1, U_2, \dots, U_{2i-2} | U_{2i-1}, U_{2i}, U_{2i+1}, U_{2i+2}}(y_1, y_2, \dots, y_n, \\ & \quad u_1, u_2, \dots, u_{2i-2} | u_{2i-1}, u_{2i}, u_{2i+1}, u_{2i+2}) \\ &\stackrel{(a)}{=} \frac{1}{4} \sum_{u_{2i+1}, u_{2i+2} \in \{0,1\}} \mathbb{P}_{Y_1, Y_2, \dots, Y_n, \tilde{U}_1, \tilde{U}_2, \dots, \tilde{U}_{2i-2} | \tilde{U}_{2i-1}, \tilde{U}_{2i}, \tilde{U}_{2i+1}, \tilde{U}_{2i+2}}(y_1, y_2, \dots, y_n, \\ & \quad \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i-1}, \tilde{u}_{2i}, \tilde{u}_{2i+1}, \tilde{u}_{2i+2}) \\ &= \frac{1}{4} \sum_{u_{2i+1}, u_{2i+2} \in \{0,1\}} \left(\mathbb{P}_{Y_1, Y_3, \dots, Y_{n-1}, \tilde{U}_1, \tilde{U}_3, \dots, \tilde{U}_{2i-3} | \tilde{U}_{2i-1}, \tilde{U}_{2i+1}}(y_1, y_3, \dots, y_{n-1}, \right. \\ & \quad \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | \tilde{u}_{2i-1}, \tilde{u}_{2i+1}) \\ & \quad \cdot \mathbb{P}_{Y_2, Y_4, \dots, Y_n, \tilde{U}_2, \tilde{U}_4, \dots, \tilde{U}_{2i-2} | \tilde{U}_{2i}, \tilde{U}_{2i+2}}(y_2, y_4, \dots, y_n, \\ & \quad \left. \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i}, \tilde{u}_{2i+2}) \right) \\ &= \frac{1}{4} \sum_{u_{2i+1}, u_{2i+2} \in \{0,1\}} \left(V_i^{(n/2)}(y_1, y_3, \dots, y_{n-1}, \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | \tilde{u}_{2i-1}, \tilde{u}_{2i+1}) \right. \\ & \quad \cdot V_i^{(n/2)}(y_2, y_4, \dots, y_n, \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i}, \tilde{u}_{2i+2}) \left. \right) \\ &= \frac{1}{4} \sum_{u_{2i+1}, u_{2i+2} \in \{0,1\}} \left(V_i^{(n/2)}(y_1, y_3, \dots, y_{n-1}, \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | u_{2i-1} + u_{2i}, u_{2i+1} + u_{2i+2}) \right. \\ & \quad \cdot V_i^{(n/2)}(y_2, y_4, \dots, y_n, \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | u_{2i}, u_{2i+2}) \left. \right) \\ &= (V_i^{(n/2)})^\nabla(y_1, y_2, \dots, y_n, \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2} | u_{2i-1}, u_{2i}), \end{aligned}$$

where $\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i+2}$ in equality (a) are defined as $\tilde{u}_{2j-1} = u_{2j-1} + u_{2j}$ and $\tilde{u}_{2j} = u_{2j}$ for $1 \leq j \leq i+1$. Finally, by noting that there is a one-to-one mapping between $y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i-2}$ in the first line and $y_1, y_2, \dots, y_n, \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2}$ in the last line, we conclude that $V_{2i-1}^{(n)} = (V_i^{(n/2)})^\nabla$. The

proofs of $V_{2i}^{(n)} = (V_i^{(n/2)})^\diamond$ and $V_{2i+1}^{(n)} = (V_i^{(n/2)})^\Delta$ are similar. We include them here for the sake of completeness.

$$\begin{aligned}
& V_{2i}^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i-1} | u_{2i}, u_{2i+1}) \\
&= \mathbb{P}_{Y_1, Y_2, \dots, Y_n, U_1, U_2, \dots, U_{2i-1} | U_{2i}, U_{2i+1}}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i-1} | u_{2i}, u_{2i+1}) \\
&= \frac{1}{4} \sum_{u_{2i+2} \in \{0,1\}} \mathbb{P}_{Y_1, Y_2, \dots, Y_n, U_1, U_2, \dots, U_{2i-2} | U_{2i-1}, U_{2i}, U_{2i+1}, U_{2i+2}}(y_1, y_2, \dots, y_n, \\
&\quad u_1, u_2, \dots, u_{2i-2} | u_{2i-1}, u_{2i}, u_{2i+1}, u_{2i+2}) \\
&\stackrel{(a)}{=} \frac{1}{4} \sum_{u_{2i+2} \in \{0,1\}} \mathbb{P}_{Y_1, Y_2, \dots, Y_n, \tilde{U}_1, \tilde{U}_2, \dots, \tilde{U}_{2i-2} | \tilde{U}_{2i-1}, \tilde{U}_{2i}, \tilde{U}_{2i+1}, \tilde{U}_{2i+2}}(y_1, y_2, \dots, y_n, \\
&\quad \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i-1}, \tilde{u}_{2i}, \tilde{u}_{2i+1}, \tilde{u}_{2i+2}) \\
&= \frac{1}{4} \sum_{u_{2i+2} \in \{0,1\}} \left(\mathbb{P}_{Y_1, Y_3, \dots, Y_{n-1}, \tilde{U}_1, \tilde{U}_3, \dots, \tilde{U}_{2i-3} | \tilde{U}_{2i-1}, \tilde{U}_{2i+1}}(y_1, y_3, \dots, y_{n-1}, \right. \\
&\quad \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | \tilde{u}_{2i-1}, \tilde{u}_{2i+1}) \\
&\quad \cdot \mathbb{P}_{Y_2, Y_4, \dots, Y_n, \tilde{U}_2, \tilde{U}_4, \dots, \tilde{U}_{2i-2} | \tilde{U}_{2i}, \tilde{U}_{2i+2}}(y_2, y_4, \dots, y_n, \\
&\quad \left. \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i}, \tilde{u}_{2i+2}) \right) \\
&= \frac{1}{4} \sum_{u_{2i+2} \in \{0,1\}} \left(V_i^{(n/2)}(y_1, y_3, \dots, y_{n-1}, \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | \tilde{u}_{2i-1}, \tilde{u}_{2i+1}) \right. \\
&\quad \left. \cdot V_i^{(n/2)}(y_2, y_4, \dots, y_n, \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i}, \tilde{u}_{2i+2}) \right) \\
&= \frac{1}{4} \sum_{u_{2i+2} \in \{0,1\}} \left(V_i^{(n/2)}(y_1, y_3, \dots, y_{n-1}, \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | u_{2i-1} + u_{2i}, u_{2i+1} + u_{2i+2}) \right. \\
&\quad \left. \cdot V_i^{(n/2)}(y_2, y_4, \dots, y_n, \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | u_{2i}, u_{2i+2}) \right) \\
&= (V_i^{(n/2)})^\diamond(y_1, y_2, \dots, y_n, \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2}, u_{2i-1} | u_{2i}, u_{2i+1}),
\end{aligned}$$

where $\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i+2}$ in equality (a) are defined the same way as above. This proves $V_{2i}^{(n)} = (V_i^{(n/2)})^\diamond$.

$$\begin{aligned}
& V_{2i+1}^{(n)}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i} | u_{2i+1}, u_{2i+2}) \\
&= \mathbb{P}_{Y_1, Y_2, \dots, Y_n, U_1, U_2, \dots, U_{2i} | U_{2i+1}, U_{2i+2}}(y_1, y_2, \dots, y_n, u_1, u_2, \dots, u_{2i} | u_{2i+1}, u_{2i+2}) \\
&= \frac{1}{4} \mathbb{P}_{Y_1, Y_2, \dots, Y_n, U_1, U_2, \dots, U_{2i-2} | U_{2i-1}, U_{2i}, U_{2i+1}, U_{2i+2}}(y_1, y_2, \dots, y_n, \\
&\quad u_1, u_2, \dots, u_{2i-2} | u_{2i-1}, u_{2i}, u_{2i+1}, u_{2i+2}) \\
&\stackrel{(a)}{=} \frac{1}{4} \mathbb{P}_{Y_1, Y_2, \dots, Y_n, \tilde{U}_1, \tilde{U}_2, \dots, \tilde{U}_{2i-2} | \tilde{U}_{2i-1}, \tilde{U}_{2i}, \tilde{U}_{2i+1}, \tilde{U}_{2i+2}}(y_1, y_2, \dots, y_n, \\
&\quad \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i-1}, \tilde{u}_{2i}, \tilde{u}_{2i+1}, \tilde{u}_{2i+2}) \\
&= \frac{1}{4} \mathbb{P}_{Y_1, Y_3, \dots, Y_{n-1}, \tilde{U}_1, \tilde{U}_3, \dots, \tilde{U}_{2i-3} | \tilde{U}_{2i-1}, \tilde{U}_{2i+1}}(y_1, y_3, \dots, y_{n-1}, \\
&\quad \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | \tilde{u}_{2i-1}, \tilde{u}_{2i+1}) \\
&\quad \cdot \mathbb{P}_{Y_2, Y_4, \dots, Y_n, \tilde{U}_2, \tilde{U}_4, \dots, \tilde{U}_{2i-2} | \tilde{U}_{2i}, \tilde{U}_{2i+2}}(y_2, y_4, \dots, y_n, \\
&\quad \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i}, \tilde{u}_{2i+2}) \\
&= \frac{1}{4} V_i^{(n/2)}(y_1, y_3, \dots, y_{n-1}, \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | \tilde{u}_{2i-1}, \tilde{u}_{2i+1}) \\
&\quad \cdot V_i^{(n/2)}(y_2, y_4, \dots, y_n, \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | \tilde{u}_{2i}, \tilde{u}_{2i+2})
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{4} V_i^{(n/2)}(y_1, y_3, \dots, y_{n-1}, \tilde{u}_1, \tilde{u}_3, \dots, \tilde{u}_{2i-3} | u_{2i-1} + u_{2i}, u_{2i+1} + u_{2i+2}) \\
&\quad \cdot V_i^{(n/2)}(y_2, y_4, \dots, y_n, \tilde{u}_2, \tilde{u}_4, \dots, \tilde{u}_{2i-2} | u_{2i}, u_{2i+2}) \\
&= (V_i^{(n/2)})^\Delta(y_1, y_2, \dots, y_n, \tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i-2}, u_{2i-1}, u_{2i} | u_{2i+1}, u_{2i+2}),
\end{aligned}$$

where $\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_{2i+2}$ in equality (a) are defined the same way as above. This proves $V_{2i+1}^{(n)} = (V_i^{(n/2)})^\Delta$ and completes the proof of Lemma 1. \square

REFERENCES

- [1] G. Li, M. Ye, and S. Hu, "Adjacent-bits-swapped polar codes: A new code construction to speed up polarization," in *2022 IEEE International Symposium on Information Theory (ISIT)*, 2022, pp. 2142–2147.
- [2] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.
- [3] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, 1954.
- [4] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection," *Transactions of the IRE professional group on electronic computers*, no. 3, pp. 6–12, 1954.
- [5] S. Kudekar, S. Kumar, M. Mondelli, H. D. Pfister, E. Şaşıoğlu, and R. Urbanke, "Reed–Muller codes achieve capacity on erasure channels," *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4298–4316, 2017.
- [6] G. Reeves and H. D. Pfister, "Reed-Muller codes achieve capacity on BMS channels," 2021, arXiv:2110.14631.
- [7] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "From polar to Reed-Muller codes: A technique to improve the finite-length performance," *IEEE Transactions on Communications*, vol. 62, no. 9, pp. 3084–3091, 2014.
- [8] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed-Muller codes," *IEEE Transactions on Information Theory*, vol. 66, no. 8, pp. 4948–4965, 2020.
- [9] S. H. Hassani, K. Alishahi, and R. Urbanke, "Finite-length scaling for polar codes," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5875–5898, 2014.
- [10] H. Hassani, S. Kudekar, O. Ordentlich, Y. Polyanskiy, and R. Urbanke, "Almost optimal scaling of Reed-Muller codes on BEC and BSC channels," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 311–315.
- [11] E. Abbe and M. Ye, "Reed-Muller codes polarize," *IEEE Transactions on Information Theory*, vol. 66, no. 12, pp. 7311–7332, 2020.
- [12] I. Dumer and K. Shabunov, "Soft-decision decoding of Reed-Muller codes: Recursive lists," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1260–1266, 2006.
- [13] M. Lian, C. Häger, and H. D. Pfister, "Decoding Reed–Muller codes using redundant code constraints," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 42–47.
- [14] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Automorphism ensemble decoding of Reed–Muller codes," *IEEE Transactions on Communications*, vol. 69, no. 10, pp. 6424–6438, 2021.
- [15] V. Guruswami and P. Xia, "Polar codes: Speed of polarization and polynomial gap to capacity," *IEEE Transactions on Information Theory*, vol. 61, no. 1, pp. 3–16, 2015.
- [16] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "Scaling exponent of list decoders with applications to polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 9, pp. 4838–4851, 2015.
- [17] —, "Unified scaling of polar codes: Error exponent, scaling exponent, moderate deviations, and error floors," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 6698–6712, 2016.
- [18] S. Buzaglo, A. Fazeli, P. H. Siegel, V. Taranalli, and A. Vardy, "Permuted successive cancellation decoding for polar codes," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 2618–2622.
- [19] —, "On efficient decoding of polar codes with large kernels," in *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2017, pp. 1–6.
- [20] M. Ye and A. Barg, "Polar codes using dynamic kernels," in *2015 IEEE International Symposium on Information Theory (ISIT)*, 2015, pp. 231–235.
- [21] A. Fazeli, H. Hassani, M. Mondelli, and A. Vardy, "Binary linear codes with optimal scaling: Polar codes with large kernels," *IEEE Transactions on Information Theory*, vol. 67, no. 9, pp. 5693–5710, 2021.
- [22] H.-P. Wang and I. M. Duursma, "Polar codes' simplicity, random codes' durability," *IEEE Transactions on Information Theory*, vol. 67, no. 3, pp. 1478–1508, 2021.
- [23] V. Guruswami, A. Riazanov, and M. Ye, "Arıkan meets Shannon: Polar codes with near-optimal convergence to channel capacity," *IEEE Transactions on Information Theory*, vol. 68, no. 5, pp. 2877–2919, 2022.
- [24] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, 2013.
- [25] I. Tal, A. Sharov, and A. Vardy, "Constructing polar codes for non-binary alphabets and MACs," in *2012 IEEE International Symposium on Information Theory Proceedings*, 2012, pp. 2132–2136.
- [26] U. Pereg and I. Tal, "Channel upgradation for non-binary input alphabets and MACs," *IEEE Transactions on Information Theory*, vol. 63, no. 3, pp. 1410–1424, 2017.
- [27] T. C. Gulcu, M. Ye, and A. Barg, "Construction of polar codes for arbitrary discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 309–321, 2018.

- [28] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.
- [29] I. Duursma, R. Gabrys, V. Guruswami, T.-C. Lin, and H.-P. Wang, "Accelerating polarization via alphabet extension," 2022, arXiv:2207.04522.