A Dynamic Programming Method to Construct Polar Codes with Improved Performance

Guodong Li  Min Ye  Sihuang Hu

Abstract

In the standard polar code construction, the message vector \((U_0, U_1, \ldots, U_{n-1})\) is divided into information bits and frozen bits according to the reliability of each \(U_i\) given \((U_0, U_1, \ldots, U_{i-1})\) and all the channel outputs. While this reliability function is the most suitable measure to choose information bits under the Successive Cancellation (SC) decoder, there is a mismatch between this reliability function and the Successive Cancellation List (SCL) decoder because the SCL decoder also makes use of the information from the future frozen bits.

We propose a Dynamic Programming (DP) construction of polar codes to resolve this mismatch. Our DP construction chooses different sets of information bits for different list sizes in order to optimize the performance of the constructed code under the SCL decoder. Simulation results show that our DP-polar codes consistently demonstrate 0.3–1dB improvement over the standard polar codes under the SCL decoder with list size 32 for various choices of code lengths and code rates.

I. INTRODUCTION

In Arıkan’s seminal work [1], Polar codes were proposed and shown to achieve capacity of any binary-input memoryless symmetric (BMS) channel under the Successive Cancellation (SC) decoder. Since then, various code construction algorithms and decoders have been proposed for polar codes. Notable polar code construction algorithms include the channel upgrading/degrading method [2], the density evolution method [3], [4], and the Monte Carlo method proposed by Arıkan in his original paper [1]. On the decoding side, the Successive Cancellation List (SCL) decoder [5] achieves almost the same performance as the Maximum Likelihood (ML) decoder when the list size is 32. The introduction of CRC (Cyclic Redundancy Check) in the code construction and the SCL decoder further reduces the decoding error probability of polar codes [5], [6].

In the polar coding scheme, the message vector \((U_0, U_1, \ldots, U_{n-1})\) of length \(n = 2^m\) is encoded into the codeword \((X_0, X_1, \ldots, X_{n-1}) = (U_0, U_1, \ldots, U_{n-1}) \circ \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes m}\), where \(\otimes\) is the Kronecker product. Each \(X_i\) is then transmitted through a BMS channel \(W\), and the corresponding channel output is denoted as \(Y_i\). Commonly-used polar code constructions [1–4] consist of two steps. The first step is to calculate the reliability of each \(U_i\) given \((U_0, U_1, \ldots, U_{i-1})\) and \((Y_0, Y_1, \ldots, Y_{n-1})\), e.g., the conditional entropy [6] \(H(U_i|U_0, U_1, \ldots, U_{i-1}, Y_0, Y_1, \ldots, Y_{n-1})\). Then the next step is to divide \((U_0, U_1, \ldots, U_{n-1})\) into information bits and frozen bits. More precisely, when constructing an \((n, k)\) polar code, we choose top \(k\) bits with the highest reliability in \((U_0, U_1, \ldots, U_{n-1})\) as the information bits. The other \(n-k\) bits are the frozen bits, and we set their values to be 0. While \(H(U_i|U_0, U_1, \ldots, U_{i-1}, Y_0, Y_1, \ldots, Y_{n-1})\) serves as the most suitable measure to choose information bits under the SC decoder, there is a mismatch...
between this reliability function and the SCL decoder. This is because the SC decoder only makes use of the previous frozen bits and the previously decoded information bits while the SCL decoder also makes use of the future frozen bits. Moreover, the SCL decoder is able to use information from more future frozen bits when it has a larger list size. In fact, when the list size is larger than \(2^k\), the SCL decoder is the same as the ML decoder. In this case, it can make use of all the future frozen bits to decode the current information bit. Therefore, in order to accurately measure the reliability of \(U_i\) in the SCL decoder, one needs to estimate the conditional entropy of \(U_i\) given both the previous bits and some of the future frozen bits, where the number of future frozen bits here increases with the list size in the SCL decoder. However, such an estimate is difficult to obtain because the number of future frozen bits whose information can be used when decoding the current bit \(U_i\) in the SCL decoder changes with both the list size and the index of \(U_i\). As a consequence, we can not follow the two-step procedure in the standard polar code constructions [1]–[4]. Instead, we propose a new construction method based on dynamic programming to obtain polar codes with better performance under the SCL decoder.

Our new construction makes use of the Plotkin \((u, u + v)\) decomposition of polar codes. More precisely, for an \((n, k)\) polar code, the vector \(u\) in the Plotkin decomposition is a codeword of a length-\(n/2\) polar code, and the vector \(v\) is a codeword of another length-\(n/2\) polar code. The dimensions of these two length-\(n/2\) polar codes sum up to \(k\). Given the list size \(L\), our code construction algorithm produces the “minus” array \(\text{minus}_L(n, k) : n = 2^m, 1 \leq m \leq 10, 0 \leq k \leq n\), where \(\text{minus}_L(n, k)\) is the number of information bits assigned to the \(v\) branch when the code length is \(n\) and code dimension is \(k\). In other words, when constructing an \((n, k)\) polar code for the SCL decoder with list size \(L\), the \(v\) branch is an \((n/2, \text{minus}_L(n, k))\) polar code, and the \(u\) branch is an \((n/2, k - \text{minus}_L(n, k))\) polar code. By recursively applying the Plotkin decomposition, the “minus” array allows us to construct polar codes with code length \(n \leq 1024\) and arbitrary code dimension.

The core of our code construction is to calculate the “minus” array, and this is done by a dynamic programming algorithm. We calculate \(\text{minus}_L(n, k)\) from small values of \(n\) to large values of \(n\). Therefore, when calculating \(\text{minus}_L(n, k)\), we already know all the values of \(\{\text{minus}_L(n', k') : n' < n, 0 \leq k' \leq n\}\), which allows us to construct polar codes with length \(n/2\) and arbitrary code dimension. In order to find the value of \(\text{minus}_L(n, k)\), we test all possible choices \(k^-\) between \(\max(0, k - n/2)\) and \(\min(k, n/2)\). When testing the performance of \(\text{minus}_L(n, k) = k^-\), we use the minus array \(\{\text{minus}_L(n', k') : n' < n, 0 \leq k' \leq n\}\) to construct the two length-\(n/2\) polar codes in the \(u\) branch and the \(v\) branch, and then we use the \((u, u + v)\) construction to combine these two length-\(n/2\) polar codes into an \((n, k)\) polar code. The performance of the choice \(\text{minus}_L(n, k) = k^-\) is measured by the decoding error probability under the SCL decoder with list size \(L\) when transmitting the above \((n, k)\) polar code over a properly chosen testing channel. The decoding error probability is estimated through Monte Carlo simulations, and the value of \(\text{minus}_L(n, k)\) is chosen to minimize the decoding error probability. In this way, our new polar code construction is specifically optimized for the SCL decoder with a given list size.

We conduct extensive simulations to compare the performance of our DP-polar codes and the standard polar codes over the binary-input AWGN channel. When we do not use CRC on either code construction, our DP-polar codes consistently demonstrate 0.3–1dB improvement over the standard polar codes under the SCL decoder with list size 32 for various choices of code rates and code length. When we use CRC on both code constructions, the performance of DP-polar code is similar to the standard polar code construction.

In addition to the improvement on the decoding error probability, our new code construction reveals an unexpected connection between polar codes and Reed-Muller (RM) codes. As mentioned above, our code construction varies with the list size of the SCL decoder. We find that when we set the list size to be 1, the DP-polar codes are almost the same as the standard polar codes. On the other hand, if we set the list size to be very large, then the DP-polar codes are almost the same as the RM codes. Since the

\(^2\)Note that our construction algorithm works for any code length if there is enough computing resource. In this paper we limit ourselves to \(n \leq 1024\) because the construction for larger code length requires more computing resource than we can afford.
SCL decoder is almost the same as the ML decoder when the list size is very large, this suggests that RM codes have better performance than (standard) polar codes under the ML decoder.

The rest of this paper is organized as follows: In the next section, we introduce the notation and present necessary background on polar codes and RM codes. In Section III, we explain the mismatch between the standard polar code construction and the SCL decoder in detail. In Section IV, we present our DP-polar code construction. Finally, the simulation results are given in Section V.

II. BACKGROUND ON POLAR CODES AND REED-MULLER CODES

Let $W : \{0, 1\} \to \mathcal{Y}$ be a binary-input discrete memoryless channel with transition probabilities $\{W(y|x) : x \in \{0, 1\}, y \in \mathcal{Y}\}$. We say that $W$ is a binary-input memoryless symmetric (BMS) channel if there is a permutation $\pi$ on the output alphabet $\mathcal{Y}$ such that i) $\pi^{-1} = \pi$ and ii) $W(y|1) = W(\pi(y)|0)$ for all $y \in \mathcal{Y}$.

The basic $2 \times 2$ polar transform is illustrated in the figure above: Let $U_0, U_1$ be two i.i.d. Bernoulli-$1/2$ random variables. Let $X_0 = U_0 + U_1$ and $X_1 = U_1$ be transmitted through two independent copies of $W$, and denote the channel outputs as $Y_0$ and $Y_1$, respectively. In this way, two copies of $W$ are transformed into two synthetic channels $W^- : U_0 \to Y_0, Y_1$ and $W^+ : U_1 \to U_0, Y_0, Y_1$ defined as follows:

$$W^-(y_0, y_1|u_0) = \frac{1}{2} \sum_{u_1 \in \{0, 1\}} W(y_0|u_0 + u_1)W(y_1|u_1) \quad \text{for} \quad u_0, y_0, y_1 \in \mathcal{Y},$$

$$W^+(u_0, y_0, y_1|u_1) = \frac{1}{2} W(y_0|u_0 + u_1)W(y_1|u_1) \quad \text{for} \quad u_0, u_1 \in \{0, 1\} \text{ and } y_0, y_1 \in \mathcal{Y}. \quad (1)$$

In order to obtain polar codes with code length $n = 2^m$, we apply the $m$-step polar transform $\left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 1 \end{array}\right]^\otimes m$. More precisely, let $(U_0, U_1, \ldots, U_{n-1})$ be the message vector and let

$$(X_0, X_1, \ldots, X_{n-1}) = (U_0, U_1, \ldots, U_{n-1}) \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 1 \end{array}\right]^\otimes m$$

be the codeword. Let $(Y_0, Y_1, \ldots, Y_{n-1})$ be the channel output vector after transmitting $(X_0, X_1, \ldots, X_{n-1})$ through $n$ independent copies of $W$. It can be shown that the set of synthetic channels $\{W_i : U_i \to U_0, \ldots, U_{i-1}, Y_0, \ldots, Y_{n-1}\}_{i=0}^{n-1}$ has a one-to-one correspondence with $\{W^s : s \in \{+,-\}^m\}$, the set consisting of the $2^m$ synthetic channels obtained from $m$ steps of polar transforms to the channel $W$. When $m \to \infty$, it was shown in [1] that almost all the $2^m$ synthetic channels are either noiseless or completely noisy. In the polar code construction, the bits corresponding to the noiseless synthetic channels are the information bits, and the ones corresponding to the completely noisy synthetic channels are the frozen bits. The information bits carry the message while the frozen bits are set to some fixed value, e.g., 0.
Fig. 1: The (16, 11) polar code constructed for the Binary-Input AWGN channel $W$ with $E_b/N_0 = 2$ dB. Each blue $U_i$ is a frozen bit, and it can be viewed as a $(1, 0)$ polar code; each red $U_i$ is an information bit, and it can be viewed as a $(1, 1)$ polar code. In the first layer of polar transforms, 16 length-1 polar codes are divided into 8 pairs, and each pair is transformed into a length-2 polar code. Then in the second layer, eight length-2 polar codes are transformed into four polar codes with length 4. This process continues until we eventually combine two length-8 polar codes to obtain the (16, 11) polar code.
Fig. 1 shows a concrete example of polar code construction for \( m = 4 \). The 16 synthetic channels in this figure are \( W_i : U_i \to U_0, \ldots, U_{i-1}, Y_0, \ldots, Y_{15} \) for \( 0 \leq i \leq 15 \). For every \( 0 \leq i \leq 15 \), let \( s(i) \in \{+,-\}^4 \) be the sequence obtained by replacing “0” with “−” and replacing “1” with “+” in the 4-digit binary expansion of \( i \). For example, \( s(1) = ---+, \) and \( s(6) = +--+ \). The synthetic channel \( W_i \) is the same as \( W^{s(i)} \) for every \( 0 \leq i \leq 15 \), where \( W^{s(i)} \) is obtained by recursively applying the definition in (1). This relation allows us to calculate a tight approximation of the channel capacity of each synthetic channel \( W_i \) using the method in [2]. The capacity of each \( W_i \) is then used to decide whether \( U_i \) is an information bit or a frozen bit. More precisely, let \( \mathcal{A} \) be the index set of the information bits, i.e., \( \{U_i : i \in \mathcal{A}\} \) are the information bits, and \( |\mathcal{A}| \) is equal to the code dimension \( k \). In the standard construction of an \( (n = 2^m, k) \) polar code for a given BMS channel \( W \), the set \( \mathcal{A} \) is chosen to satisfy that the \( k \) synthetic channels \( \{W_i : i \in \mathcal{A}\} \) have the largest channel capacity among all the \( n \) synthetic channels.

The example in Fig. 1 is an \( (n = 16, k = 11) \) polar code constructed for the Binary-Input AWGN channel \( W \) with \( E_b/N_0 = 2 \) dB. In this example, \( \mathcal{A} = \{3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15\} \). On the left side of the figure, each frozen bit can be viewed as a \( (1,0) \) polar code, i.e., a polar code with length 1 and dimension 0. Similarly, each information bit can be viewed as a \( (1,1) \) polar code. In the first layer of polar transforms, two frozen bits are transformed into a \( (2,0) \) polar code, e.g., \( (U_0, U_1) \) is transformed into \( (X^{(1)}_0, X^{(1)}_1) \), which is a \( (2,0) \) polar code; one frozen bit and one information bit are transformed into a \( (2,1) \) polar code, e.g., \( (U_2, U_3) \) is transformed into \( (X^{(1)}_2, X^{(1)}_3) \); two information bits are transformed into a \( (2,2) \) polar code, e.g., \( (U_6, U_7) \) is transformed into \( (X^{(1)}_6, X^{(1)}_7) \). In the second layer of polar transforms, two length-2 polar codes are transformed into a length-4 polar code. For example, the \( (2,1) \) polar code \( (X^{(1)}_4, X^{(1)}_5) \) and the \( (2,2) \) polar code \( (X^{(1)}_6, X^{(1)}_7) \) are transformed into the \( (4,3) \) polar code \( (X^{(2)}_4, X^{(2)}_5, X^{(2)}_6, X^{(2)}_7) \). In the next layer, two length-4 polar codes are further transformed into a length-8 polar codes, e.g., the \( (4,1) \) polar code \( (X^{(2)}_0, X^{(2)}_1, X^{(2)}_2, X^{(2)}_3) \) and the \( (4,3) \) polar code \( (X^{(2)}_4, X^{(2)}_5, X^{(2)}_6, X^{(2)}_7) \) are transformed into the \( (8,4) \) polar code \( (X^{(3)}_0, X^{(3)}_1, \ldots, X^{(3)}_7) \). Finally, in the last layer, the \( (8,4) \) polar code \( (X^{(3)}_0, X^{(3)}_1, \ldots, X^{(3)}_7) \) and the \( (8,7) \) polar code \( (X^{(3)}_8, X^{(3)}_9, \ldots, X^{(3)}_{15}) \) are transformed into the \( (16,11) \) polar code \( (X_0, X_1, \ldots, X_{15}) \).

A. Successive Cancellation decoder of polar codes

The Successive Cancellation (SC) decoder of polar codes works in a recursive way. When decoding a length-\( n \) polar code, the SC decoder decomposes it into two length-\( n/2 \) polar codes and decodes these two codes one by one. For example, the \( (16,11) \) polar code in Fig. 1 can be decomposed into the \( (8,4) \) polar code \( (X^{(3)}_0, X^{(3)}_1, \ldots, X^{(3)}_7) \) and the \( (8,7) \) polar code \( (X^{(3)}_8, X^{(3)}_9, \ldots, X^{(3)}_{15}) \). The SC decoder first uses the channel output \( (Y_0, Y_1, \ldots, Y_{15}) \) to decode the \( (8,4) \) polar code \( (X^{(3)}_0, X^{(3)}_1, \ldots, X^{(3)}_7) \). Let us denote the decoding result as \( (\hat{X}^{(3)}_0, \hat{X}^{(3)}_1, \ldots, \hat{X}^{(3)}_7) \). Then the SC decoder uses both the channel output \( (Y_0, Y_1, \ldots, Y_{15}) \) and the previous decoding result \( (\hat{X}^{(3)}_0, \hat{X}^{(3)}_1, \ldots, \hat{X}^{(3)}_7) \) to decode the \( (8,7) \) polar code \( (X^{(3)}_8, X^{(3)}_9, \ldots, X^{(3)}_{15}) \), whose decoding result is denoted as \( (\hat{X}^{(3)}_8, \hat{X}^{(3)}_9, \ldots, \hat{X}^{(3)}_{15}) \).

Finally, the SC decoder applies the last layer of polar transform to \( (\hat{X}^{(3)}_0, \hat{X}^{(3)}_1, \ldots, \hat{X}^{(3)}_7) \) and \( (\hat{X}^{(3)}_8, \hat{X}^{(3)}_9, \ldots, \hat{X}^{(3)}_{15}) \) to obtain the final decoding result \( (\hat{X}_0, \hat{X}_1, \ldots, \hat{X}_{15}) \). This describes the outer layer of the SC decoder. In the next layer, when decoding the two length-8 polar codes, the SC decoder follows the same recursive structure. More precisely, when decoding the \( (8,4) \) polar code \( (X^{(3)}_0, X^{(3)}_1, \ldots, X^{(3)}_7) \), the SC decoder first decodes the \( (4,1) \) polar code \( (X^{(2)}_0, X^{(2)}_1, X^{(2)}_2, X^{(2)}_3) \) and then decodes the \( (4,3) \) polar code \( (X^{(2)}_4, X^{(2)}_5, X^{(2)}_6, X^{(2)}_7) \). Similarly, when decoding the \( (8,7) \) polar code \( (X^{(3)}_8, X^{(3)}_9, \ldots, X^{(3)}_{15}) \), the SC decoder first decodes the \( (4,3) \) polar code \( (X^{(3)}_8, X^{(3)}_9, X^{(3)}_{10}, X^{(3)}_{11}) \) and then decodes the \( (4,4) \) polar code \( (X^{(3)}_{12}, X^{(3)}_{13}, X^{(3)}_{14}, X^{(3)}_{15}) \). This recursive procedure continues until the SC decoder reaches the length-1 polar codes, i.e., the frozen bits and the information bits \( U_0, U_1, \ldots, U_{15} \). To summarize, in the
tree diagram in Fig. [1] the decoding procedure of the SC decoder goes from right to left and from top to bottom.

For $0 \leq i \leq 7$, the channel mapping from $X_i^{(3)}$ to $(Y_0, Y_1, \ldots, Y_{15})$ is $W^{-}$ because only $Y_i$ and $Y_{i+8}$ depend on the value of $X_i^{(3)}$. For $8 \leq i \leq 15$, the channel mapping from $X_i^{(3)}$ to $(X_0^{(3)}, X_1^{(3)}, \ldots, X_7^{(3)}, Y_0, Y_1, \ldots, Y_{15})$ is $W^{+}$. Therefore, the $(8, 4)$ polar code $(X_0^{(3)}, X_1^{(3)}, \ldots, X_7^{(3)})$ in the top branch is constructed for $W^{-}$, and the $(8, 7)$ polar code $(X_8^{(3)}, X_9^{(3)}, \ldots, X_{15}^{(3)})$ in the bottom branch is constructed for $W^{+}$. In fact, every top branch in the tree diagram corresponds to the “minus” transform on the channel, and every bottom branch corresponds to the “plus” transform. For example, the $(4, 1)$ polar code $(X_0^{(2)}, X_1^{(2)}, X_2^{(2)}, X_3^{(2)})$ is constructed for $W^{--}$, and the $(2, 1)$ polar code $(X_2^{(1)}, X_3^{(1)})$ is constructed for $W^{-}$. In this paper, we will use the terms “top branch” and “minus branch” interchangeably, and we also use the terms “bottom branch” and “plus branch” interchangeably.

As a final remark, we note that the binary tree representation in Fig. [1] has a natural connection with the Plotkin $(u, u + v)$ decomposition: The top branch in the binary tree corresponds to the component $v$, and the bottom branch corresponds to the component $u$.

B. Reed-Muller codes

The generator matrices of both polar codes and RM codes are submatrices of $G_m := \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes^m$. Polar codes select row vectors corresponding to noiseless synthetic channels while RM codes select row vectors with the largest Hamming weight. More precisely, we label the row indices of $G_m$ from 0 to $2^m - 1$. For $0 \leq i \leq 2^m - 1$, we use $\text{wt}(i)$ to denote the Hamming weight of the $m$-digit binary expansion of $i$. Then the Hamming weight of the $i$th row of $G_m$ is $2^{\text{wt}(i)}$. The generator matrix of the RM code with parameters $m$ and $r$ consists of all the row vectors of $G_m$ whose indices satisfy $\text{wt}(i) \geq m - r$.

Note that originally RM codes were defined using multivariate polynomials with binary coefficients [7], [8], but for the purpose of this paper we will stick to the definition described above. Readers may consult a recent survey [9] on RM codes to see why these two definitions are equivalent.

III. MISMATCH BETWEEN POLAR CODE CONSTRUCTION AND THE SCL DECODER

Following the notation in Fig. [1] for a length-$n$ polar code, we use $(U_0, U_1, \ldots, U_{n-1})$ to denote the vector containing the information bits and the frozen bits. We use $(X_0, X_1, \ldots, X_{n-1})$ and $(Y_0, Y_1, \ldots, Y_{n-1})$ to denote the codeword and the channel output vector, respectively. Given a BMS channel $W$, the code length $n$ and the code dimension $k$, polar code construction amounts to finding which $k$ bits in $(U_0, U_1, \ldots, U_{n-1})$ are the information bits. In the standard polar code construction, e.g., the construction described in [2], we calculate a tight approximation of the conditional entropy

$$H_i := H(U_i | U_0, U_1, \ldots, U_{i-1}, Y_0, Y_1, \ldots, Y_{n-1})$$

and then choose the $k$ information bits as the ones corresponding to the smallest values of $H_i$’s. This code construction works perfectly for the SC decoder because the SC decoder only makes use of the previous message bits, i.e., $U_0, U_1, \ldots, U_{i-1}$. However, there is a mismatch between this code construction and the SCL decoder because the list decoder also makes use of the information from the future frozen bits. To see this, let us first recall how the SCL decoder works. Similarly to the SC decoder, the SCL decoder also decodes from $U_0$ to $U_{n-1}$ one by one. The difference is that the SC decoder only keeps one candidate of the decoding results of previous bits while the SCL decoder keeps a list of multiple candidates. When reaching an information bit, the SCL decoder expand the current list size by a factor of 2 to append both possible values (i.e., 0 and 1) of this information bit at the end of each candidate. A pruning procedure according to the likelihood of the candidates will be activated if the current list size becomes too large after the expansion. When reaching a frozen bit, the likelihood of each candidate in

\footnote{Conditional entropy is not the only choice. We can also use other metrics to measure the uncertainty of $U_i$ given $U_0, U_1, \ldots, U_{i-1}, Y_0, Y_1, \ldots, Y_{n-1}$, e.g., the Bhattacharyya parameter.}
the list is updated based on the conditional probability of that frozen bit given the values of the previous bits and the channel outputs. This can penalize some candidates by a considerable amount if the values of the previously decoded bits in the candidate do not agree with the value of the frozen bit. In this way, the frozen bit serves as a “soft parity check” for the previously decoded information bits, or in other words, the decoding decision on each information bit relies not only on the previous bits but also on the future frozen bits.

Let us take the following extreme case as an example. When the list size is larger than \(2^k\), the SCL decoder is the same as the Maximum Likelihood (ML) decoder. In this case, the uncertainty of each information bit under the SCL decoder is

\[
H_i^{\text{ML}} := H(U_i|\{U_j : j \in A^c\}, Y_0, Y_1, \ldots, Y_{n-1}),
\]

where \(A\) is the index set of information bits and \(A^c\) is the index set of frozen bits. Notice that for the standard polar code construction,

\[
H_i \approx H(U_i|\{U_j : j \in A^c, j < i\}, Y_0, Y_1, \ldots, Y_{n-1}),
\]

where \(H_i\) is defined in (2). This is because for the standard polar code construction, the information bits \(\{U_j : j \in A, j < i\}\) can be determined by the frozen bits \(\{U_j : j \in A^c, j < i\}\) and the channel outputs \(Y_0, Y_1, \ldots, Y_{n-1}\) with high probability. Comparing (3) with (4), we can see that \(H_i^{\text{ML}}\) is typically smaller than \(H_i\), indicating that there is a mismatch between the polar code construction and the ML decoder.

As for the normal SCL decoder whose list size is not as large as \(2^k\), the uncertainty of each information bit under the SCL decoder can be approximated by the following conditional entropy

\[
H_i^{\text{SCL}} \approx H(U_i|\{U_j : j \in A^c, j < i + \Delta_{i,L}\}, Y_0, Y_1, \ldots, Y_{n-1}),
\]

where the nonnegative integer \(\Delta_{i,L}\) depends on both the index \(i\) and the list size \(L\). By the discussion above, when \(L \geq 2^k\), we can take \(\Delta_{i,L} = n - i\). For this choice of \(\Delta_{i,L}\), we have \(\{U_j : j \in A^c, j < i + \Delta_{i,L}\} = \{U_j : j \in A^c\}\) and \(H_i^{\text{SCL}} = H_i^{\text{ML}}\). When \(L\) takes a smaller value, the dependence of \(\Delta_{i,L}\) on \(L\) is rather complicated to quantify. However, one would expect that \(\Delta_{i,L}\) is an increasing function of \(L\) because intuitively, a larger list size allows the SCL decoder to make use of information from more future frozen bits. In the extreme case of \(L = 1\), we have \(\Delta_{i,L} = 0\) and \(H_i^{\text{SCL}} \approx H_i\) because in this case the SCL decoder degenerates into the SC decoder. Now we conclude that

\[
H_i \geq H_i^{\text{SCL}} \geq H_i^{\text{ML}}.
\]

Moreover, when the list size is small, \(H_i^{\text{SCL}}\) tends to be close to \(H_i\); when the list size is large, \(H_i^{\text{SCL}}\) tends to be close to \(H_i^{\text{ML}}\). From a practical point of view, the simulation results in [5] tell us that when the list size is \(L = 32\), the performance of the SCL decoder is already very close to the ML decoder for the Binary-Input AWGN channel with \(E_b/N_0 > 1.5 \text{ dB}\). Therefore, for list size \(L = 32\), \(H_i^{\text{SCL}}\) is very close to \(H_i^{\text{ML}}\) for a wide range of channels. This further indicates that for some values of \(i\), the gap between \(H_i^{\text{SCL}}\) and \(H_i\) might not be negligible.

Next let us consider how to improve the polar code construction to compensate for the gap between \(H_i^{\text{SCL}}\) and \(H_i\). Observe that \(U_i\) with smaller index \(i\) have more future frozen bits in front of it, so intuitively, the gap \(H_i - H_i^{\text{SCL}}\) tends to be larger for smaller value of \(i\). This suggests that the reliability of \(U_i\) with relatively small index are more likely to be underestimated by the standard polar code construction. As a consequence, in the optimal polar code construction for the SCL decoder, more information bits should be assigned to the minus branch than in the standard polar code construction. In the next section, we will develop a dynamic programming algorithm to find the optimal number of information bits in the minus branch for the SCL decoder.

footnote: Recall the definition of minus branch at the end of Section II: By minus branch we simply mean \((U_0, U_1, \ldots, U_{n/2-1})\), the first half of \((U_0, U_1, \ldots, U_{n-1})\).
IV. A Dynamic Programming Algorithm to Construct Polar Codes for SCL Decoder

Similarly to RM codes and the standard polar codes, the generator matrices of our DP-polar codes are also submatrices of $G_m := \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^\otimes m$. Among all the $(n = 2^m, k)$ codes whose generator matrices consist of $k$ row vectors of $G_m$, we want to find the one with the best performance under the SCL decoder. A naive way to do so is to test the decoding error probability of all the $\binom{n}{k}$ candidates under the SCL decoder. However, the complexity of this naive method is too high for large values of $n$. Therefore, we propose the following dynamic programming method to find the best $k$ rows of $G_m$ under the SCL decoder.

Although our DP-polar code construction works for any class of communication channels, below we will focus on the construction for binary-input AWGN channel because this is the most important communication channel in practice.

A. Construct Polar Codes from the Minus Array

Our code construction depends on three parameters—the code length $n$, the code dimension $k$, and the list size $L$. Following the binary tree decomposition in Fig. 1, we use $\text{minus}_L(n, k)$ to denote the number of information bits assigned to the minus branch when we construct an $(n, k)$ polar code that is optimized for list size $L$. For a fixed value of $L$, if we know $\text{minus}_L(n, k)$ for all possible pairs of $(n, k)$, then we can construct polar codes with any code length and code dimension according to the binary tree decomposition. More precisely, constructing an $(n, k)$ polar code amounts to finding $A$, the index set of the $k$ information bits. Let $k^- = \text{minus}_L(n, k)$ and $k^+ = k - k^-$. Then we know that $k^-$ information bits are assigned to the minus branch, i.e., there are $k^-$ information bits in $(U_0, U_1, \ldots, U_{n/2-1})$. Similarly, there are $k^+$ information bits in $(U_{n/2}, U_{n/2+1}, \ldots, U_{n-1})$. Next we look at the values of $k^- = \text{minus}_L(n/2, k^-), k^+ = k^- - k^-, k^+ + k^- = \text{minus}_L(n/2, k^+), k^+ + k^+ = k^+ - k^+$. According to the binary tree decomposition, $k^-$ is the number of information bits in $(U_0, \ldots, U_{n/4-1})$; $k^- +$ is the number of information bits in $(U_{n/4}, \ldots, U_{n/2-1})$; $k^+ +$ is the number of information bits in $(U_{n/2}, \ldots, U_{3n/4-1})$; $k^+ +$ is the number of information bits in $(U_{3n/4}, \ldots, U_{n-1})$. This decomposition continues until we obtain $\{k^s : s \in \{+,-\}^m\}$, where the value of each $k^s$ indicates whether the corresponding $U_i$ is an information bit or a frozen bit. More precisely, for every $0 \leq i \leq 2^m - 1$, let $s(i) \in \{+,-\}^m$ be the sequence obtained by replacing “0” with “-” and replacing “1” with “+” in the $m$-digit binary expansion of $i$. Then $U_i$ is an information bit if $k^{s(i)} = 1$, and $U_i$ is a frozen bit if $k^{s(i)} = 0$. Note that $k^s$ is either 0 or 1 for all $s \in \{+,-\}^m$. In this way, we obtain an $(n, k)$ polar code that is optimized for the SCL decoder with list size $L$. This construction method is summarized in...
Algorithm 1: MinusConstruct\((n, k, \{\text{minus}_L(n', k') : n' \leq n, 0 \leq k' \leq n'\})\)

**Input:** code length \(n \geq 2\), code dimension \(k\), and the minus array \(\{\text{minus}_L(n', k') : n' \leq n, 0 \leq k' \leq n'\}\)  

**Output:** the index set \(A\) of the information bits

\[
\begin{align*}
1 & \quad m \leftarrow \log_2(n) \\
2 & \quad A \leftarrow \emptyset \quad \triangleright \text{Initialize } A \text{ as the empty set.} \\
3 & \quad \text{for } j = 0, 1, \ldots, m - 1 \text{ do} \\
4 & \quad \quad \text{for } s \in \{+, -\}^j \text{ do} \\
5 & \quad \quad \quad k^s - \leftarrow \text{minus}_L(n/2^j, k^s) \quad \triangleright \text{s is the empty string when } j = 0. \text{ In this case, } k^s = k. \\
6 & \quad \quad \quad k^s + \leftarrow k^s - k^{s -} \quad \triangleright \text{s is obtained from appending a “−” at the end of the string } s \\
7 & \quad \quad \text{for } i = 0, 1, 2, \ldots, n - 1 \text{ do} \\
8 & \quad \quad \quad \text{if } k^s(i) = 1 \text{ then} \\
9 & \quad \quad \quad \quad A \leftarrow A \cup \{i\} \\
10 & \quad \quad \text{return } A
\end{align*}
\]

| \(k\) | \(n\) | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 2    | 0    | 0  | 0  | 1  |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4    | 0    | 0  | 0  | 0  | 1  | 2  |    |    |    |    |    |    |    |    |    |    |    |
| 8    | 0    | 0  | 0  | 0  | 1  | 1  | 2  | 3  | 4  |    |    |    |    |    |    |    |    |
| 16   | 0    | 0  | 0  | 0  | 1  | 1  | 1  | 2  | 2  | 2  | 3  | 4  | 4  | 6  | 7  | 7  | 8  |

**TABLE I:** An example of the minus array with \(n \leq 16\) and \(0 \leq k \leq n\).

As a concrete example, we show how to construct an \((n = 16, k = 11)\) code from the minus array in Table I. The first step is to calculate \(k^−\) and \(k^+\):

\[
k^− = \text{minus}_L(16, 11) = 4, \quad k^+ = 11 - k^− = 7.
\]

The next step is to calculate \(k^{−−}, k^{−+}, k^{+−}, k^{++}\):

\[
k^{−−} = \text{minus}_L(8, k^−) = 1, \quad k^{−+} = k^− - k^{−−} = 3, \\
k^{+−} = \text{minus}_L(8, k^+) = 3, \quad k^{++} = k^+ - k^{+−} = 4.
\]

Next we calculate \(k^s, s \in \{+, −\}^3\):

\[
k^{−−−} = \text{minus}_L(4, k^{−−}) = 0, \quad k^{−−+} = k^{−−} - k^{−−−} = 1, \\
k^{−++} = \text{minus}_L(4, k^{−+}) = 1, \quad k^{−++} = k^{−+} - k^{−++} = 2, \\
k^{+++} = \text{minus}_L(4, k^{++}) = 1, \quad k^{+++} = k^{++} - k^{+++} = 2, \\
k^{−−+} = \text{minus}_L(4, k^{−−}) = 2, \quad k^{−++} = k^{−+} - k^{−++} = 2.
\]

Finally, we obtain

\[
k^{−−−} = 0, \quad k^{−−+} = 0, \quad k^{−++} = 0, \quad k^{−+++} = 1, \\
k^{−−−} = 0, \quad k^{−−+} = 1, \quad k^{−++} = 1, \quad k^{−+++} = 1, \\
k^{−−−} = 0, \quad k^{−−+} = 1, \quad k^{−++} = 1, \quad k^{−+++} = 1,
\]
so the index set of the information bits is \( A = \{3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15\} \). This completes the construction of the \((n = 16, k = 11)\) code. Note that this \((16, 11)\) code is the same as the code in Fig. 1.

### B. Obtain the minus array by dynamic programming

Let’s say we want to construct polar codes with code length up to \( N \), where \( N \) is a power of 2. In our implementation, we take \( N = 1024 \). For a given list size \( L \), we calculate the minus array \( \text{minus}_L(n, k) \) from small values of \( n \) to large values of \( n \). For each fixed value of \( n \), we calculate \( \text{minus}_L(n, k) \) from \( k = 0 \) to \( k = n \). As a consequence, when we calculate the value of \( \text{minus}_L(n, k) \), we already know \( \{\text{minus}_L(n', k') : n' \leq n/2, 0 \leq k' \leq n'\} \). It is easy to see that the range of \( \text{minus}_L(n, k) \) is

\[
\max(0, k - n/2) \leq \text{minus}_L(n, k) \leq \min(k, n/2).
\]

(6)

For each choice of \( \text{minus}_L(n, k) = k^- \), we construct the corresponding polar codes as follows: Let \( A_1 \) be the index set of information bits returned by Algorithm 1 when the inputs to the algorithm are code length \( n/2 \), code dimension \( k^- \), and the minus array \( \{\text{minus}_L(n', k') : n' \leq n/2, 0 \leq k' \leq n'\} \). Let \( A_2 \) be the index set of information bits returned by Algorithm 1 when the inputs to the algorithm are code length \( n/2 \), code dimension \( k - k^- \), and the minus array \( \{\text{minus}_L(n', k') : n' \leq n/2, 0 \leq k' \leq n'\} \). Note that both \( A_1 \) and \( A_2 \) are subsets of \( \{0, 1, 2, \ldots, n/2 - 1\} \). Next define \( A_2 + n/2 := \{i + n/2 : i \in A_2\} \) and define \( A = A_1 \cup (A_2 + n/2) \). The set \( A \) is the index set of information bits in the code construction corresponding to the choice \( \text{minus}_L(n, k) = k^- \). For each \( k^- \) between \( \max(0, k - n/2) \) and \( \min(k, n/2) \), we test the performance of its corresponding code construction under the SCL decoder with list size \( L \). The testing channel is the binary-input AWGN channel with \( E_b/N_0 = 2 \text{ dB} \). We set the value of \( \text{minus}_L(n, k) \) to be the optimal choice of \( k^- \) which minimizes the decoding error probability under the SCL decoder with list size \( L \). This dynamic programming approach is summarized in Algorithm 2 below. The function \( \text{DecodingError}(n, k, A, L) \) in line 11 of Algorithm 2 estimates the decoding error probability of the \((n, k)\) polar code under the SCL decoder with list size \( L \), where \( A \) is the index set of information bits. In our implementation, we perform \( 10^5 \) rounds of Monte Carlo simulations over the binary-input AWGN channel with \( E_b/N_0 = 2 \text{ dB} \) to estimate the decoding error probability of the SCL decoder.

**Algorithm 2: CalculateMinusArray**

`Input: upper bound \( N \) on the code length \( (N \) is a power of 2), and the list size \( L \)

`Output: the minus array \( \{\text{minus}_L(n, k) : n \leq N, 0 \leq k \leq n\} \)

1. for \( n = 2, 4, 8, \ldots, N \) do
2. for \( k = 0, 1, \ldots, n \) do
3. \( s \leftarrow \max(0, k - n/2) \)
4. \( t \leftarrow \min(k, n/2) \)
5. \( \text{err}_{\text{min}} \leftarrow 1 \) \> Initialize the minimum decoding error probability as 1
6. for \( k^- = s, s + 1, s + 2, \ldots, t \) do
7. \( A_1 \leftarrow \text{MinusConstruct}(n/2, k^-, \{\text{minus}_L(n', k') : n' \leq n/2, 0 \leq k' \leq n'\}) \)
8. \> The function \( \text{MinusConstruct} \) is defined in Algorithm 1
9. \( A_2 \leftarrow \text{MinusConstruct}(n/2, k^--k^-, \{\text{minus}_L(n', k') : n' \leq n/2, 0 \leq k' \leq n'\}) \)
10. \( A \leftarrow A_1 \cup (A_2 + n/2) \)
11. \( \text{err} \leftarrow \text{DecodingError}(n, k, A, L) \)
12. \> See the main text about how the function \( \text{DecodingError}(n, k, A, L) \) works.
13. if \( \text{err} < \text{err}_{\text{min}} \) then
14. \( \text{err}_{\text{min}} \leftarrow \text{err} \)
15. \( \text{minus}_L(n, k) \leftarrow k^- \)
16. return \( \{\text{minus}_L(n, k) : n \leq N, 0 \leq k \leq n\} \)
Remark 1. The range in (6) is too large for practical implementation. We can use the continuity of \(\operatorname{minus}_L(n, k)\) to shrink the range of search. Intuitively, we would expect that \(\operatorname{minus}_L(n, k - 1)\) is close to \(\operatorname{minus}_L(n, k)\). Since we calculate the minus array from small values of \(k\) to large values of \(k\), we already know the value of \(\operatorname{minus}_L(n, k - 1)\) when we calculate \(\operatorname{minus}_L(n, k)\). Therefore, we only need to search in a small neighborhood of \(\operatorname{minus}_L(n, k - 1)\) in order to determine the value of \(\operatorname{minus}_L(n, k)\). In our implementation, we only test 7 values in the set \(\{\operatorname{minus}_L(n, k - 1) - 2, \operatorname{minus}_L(n, k - 1) - 1, \ldots, \operatorname{minus}_L(n, k - 1) + 3, \operatorname{minus}_L(n, k - 1) + 4\}\) for the calculation of each \(\operatorname{minus}_L(n, k)\).

C. Connection with RM codes

The generator matrices of standard polar codes, DP-polar codes and RM codes are all submatrices of the same square matrix \(G_m := \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes m\). These three classes of codes pick different sets of row vectors from \(G_m\) to form their generator matrices. More precisely, standard polar codes pick the index set of information bits according to the conditional expectation \(H_i\) defined in (2); DP-polar codes pick the index set of information bits according to the minus array; and RM codes pick the index set according to the Hamming weight of the binary expansion of the row index, as described in Section II-B. This allows us to measure the similarity between these three classes of codes as follows: Let \(n = 2^m\) be the code length and let

\[
k = \left( m \atop 0 \right) + \left( m \atop 1 \right) + \left( m \atop 2 \right) + \cdots + \left( m \atop r \right)
\]

be the code dimension\(^3\), where \(r\) is some positive integer that is smaller than \(m\). For the \((n, k)\) standard polar code, we denote its index set of information bits as \(A_{\text{polar}}\). For the \((n, k)\) DP-polar code, we denote its index set of information bits as \(A_{\text{DP}}\). For the \((n, k)\) RM code, we denote its index set of information bits as \(A_{\text{RM}}\). We define the similarity between the \((n, k)\) DP-polar code and the \((n, k)\) standard polar code as

\[
S_{\text{polar}} := \frac{|A_{\text{DP}} \cap A_{\text{polar}}|}{k},
\]

where \(|A|\) denotes the size of the set \(A\). The similarity between the \((n, k)\) DP-polar code and the \((n, k)\) RM code is defined as

\[
S_{\text{RM}} := \frac{|A_{\text{DP}} \cap A_{\text{RM}}|}{k}.
\]

By definition, \(S_{\text{polar}} = 1\) implies that the \((n, k)\) DP-polar code is the same as the standard polar code. Similarly, \(S_{\text{RM}} = 1\) implies that the \((n, k)\) DP-polar code is the same as the RM code.

Since our DP-polar code construction depends on the list size \(L\), the similarity measures \(S_{\text{polar}}\) and \(S_{\text{RM}}\) are both functions of \(L\). In Tables III and IV below, we list how the values of \(S_{\text{polar}}\) and \(S_{\text{RM}}\) change with the list size \(L\). As we can see from the tables, \(S_{\text{polar}}\) always decreases with \(L\), and \(S_{\text{RM}}\) always increases with \(L\). When \(L = 1\), the DP-polar code is (almost) the same as the standard polar code. When \(L\) is large enough, e.g., when \(L = 128\), the DP-polar code is the same as the RM code. This tells us that when we increase \(L\) from 1 to infinity, the DP-polar code gradually changes from the standard polar code to the RM code.

The construction of DP-polar codes aims to optimize its performance under the SCL decoder with a given list size \(L\). When \(L\) goes to infinity, the SCL decoder is the same as the ML decoder. Therefore, the fact that DP-polar codes converge to RM codes when \(L \to \infty\) suggests that RM codes have the best performance under the ML decoder among all the codes whose generator matrices are formed of row vectors of \(G_m = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes m\). In particular, this suggests that RM codes have better performance than

\(^3\)We restrict ourselves to these specific choices of code dimension because the dimension of RM codes can only take this form.
polar codes with the same parameters under the ML decoder. Note that this observation has already been made in [10], [11]. The discussion above provides one more indirect evidence to this claim.

| $L$  | 1   | 2   | 4   | 8   | 16  | 32  | 64  | 128 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| $S_{\text{polar}}$ | 1.00 | 0.93 | 0.93 | 0.86 | 0.83 | 0.83 | 0.83 | 0.83 |
| $S_{\text{RM}}$     | 0.83 | 0.90 | 0.90 | 0.97 | 1.00 | 1.00 | 1.00 | 1.00 |

**TABLE II:** The values of $S_{\text{polar}}$ and $S_{\text{RM}}$ change with the list size $L$. Here we take $n = 128$ and $k = 29$, corresponding to the second-order RM code.

| $L$  | 1   | 2   | 4   | 8   | 16  | 32  | 64  | 128 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| $S_{\text{polar}}$ | 0.97 | 0.97 | 0.95 | 0.94 | 0.92 | 0.92 | 0.91 | 0.91 |
| $S_{\text{RM}}$     | 0.91 | 0.94 | 0.94 | 0.95 | 0.98 | 0.98 | 0.98 | 1.00 |

**TABLE III:** The values of $S_{\text{polar}}$ and $S_{\text{RM}}$ change with the list size $L$. Here we take $n = 128$ and $k = 64$, corresponding to the third-order RM code.

| $L$  | 1   | 2   | 4   | 8   | 16  | 32  | 64  | 128 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| $S_{\text{polar}}$ | 1.00 | 0.99 | 0.97 | 0.96 | 0.95 | 0.95 | 0.95 | 0.95 |
| $S_{\text{RM}}$     | 0.95 | 0.96 | 0.98 | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 |

**TABLE IV:** The values of $S_{\text{polar}}$ and $S_{\text{RM}}$ change with the list size $L$. Here we take $n = 128$ and $k = 99$, corresponding to the fourth-order RM code.

**V. SIMULATION RESULTS**

In this section, we provide simulation results to compare the performance of the DP-polar codes with the standard polar codes under the SCL decoder with list size $32$. For this choice of list size, the minus array of our DP-polar code construction is given in Appendix A. All the DP-polar codes in Fig. 2 are constructed from this minus array.

Similarly to the standard polar codes, we can also add CRC to the DP-polar code construction and the corresponding SCL decoder. In order to add a length $\ell_c$ CRC to an $(n, k)$ DP-polar code, we first construct an $(n, k + \ell_c)$ DP-polar code from the minus array. Then we divide these $k + \ell_c$ information bits into two parts: The first $k$ bits carry the message, and the last $\ell_c$ bits are (random) parity checks of the first $k$ bits. In the SCL decoding procedure, the CRC allows us to eliminate incorrect candidates in the list.

The simulation results in Fig. 2 include various choices of code length $n \in \{256, 512, 1024\}$ and code rates $k/n \in \{0.3, 0.5, 0.7\}$. In each plot, there are four curves, where the legend “DP” refers to DP-polar codes, and the legend “ST” refers to standard polar codes. Moreover, “CRC-0” in the legend means that we do not use CRC. “CRC-$\ell_c$” with a nonzero $\ell_c$ means that $\ell_c$ is the optimal CRC length that minimizes the decoding error probability at $E_b/N_0 = 2.5$ dB.

From these simulation results, we can see that when there is no CRC, our DP-polar codes consistently outperform the standard polar codes by $0.3$–$1$ dB. When we use the optimal length of CRC on both codes, the performance of DP-polar codes is similar to the standard polar codes for most choices of parameters. As a final remark, since we use exactly the same decoder for both the DP-polar codes and the standard polar codes, the decoding time of these two classes of codes are more or less the same.
Fig. 2: Comparison between DP-polar codes and standard polar codes over the binary-input AWGN channel. We use SCL decoder with list size 32 in all the plots. The legend “DP” refers to DP-polar codes, and “ST” refers to standard polar codes. “CRC-0” means that we do not use CRC. “CRC-ℓ_c” with a nonzero ℓ_c means that ℓ_c is the optimal CRC length that minimizes the decoding error probability.
**APPENDIX A**

**The minus array for** $L = 32$

| $n$ | $\Delta$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|-----------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 2   | 1         | 0 | 1 |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| 4   | 1         | 0 | 0 | 1 | 2 |   |   |   |   |   |   |    |    |    |    |    |    |
| 8   | 1         | 0 | 0 | 0 | 1 | 1 | 2 | 3 | 4 |   |   |    |    |    |    |    |    |
| 16  | 1         | 0 | 0 | 0 | 1 | 1 | 2 | 1 | 2 | 4 | 4 | 4  | 5  | 6  | 7  | 8  |    |
| 32  | 1         | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 3 | 1 | 3 | 1  | 4  | 3  | 4  | 5  | 5  |
| 64  | 1         | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 1 | 2 | 3  | 1  | 2  | 1  |    |    |
| 128 | 1         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 3 | 3 | 1  | 2  | 1  | 2  | 1  |    |
| 256 | 1         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2  | 1  | 1  | 1  | 1  | 2  |

**TABLE V:** Minus array for $L = 32, n = 2, 4, \ldots, 256$ and $1 \leq k \leq n$. The value in the table is $\text{minus}_{32}(n, k + \Delta)$. 


| $n$ | $k$ | $\Delta$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-----|-----|---------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1   |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 17  |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 33  |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 49  |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 65  |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 81  |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 97  |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 113 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 129 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 145 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 161 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 177 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 193 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 209 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 225 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 241 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 257 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 273 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 289 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 305 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 321 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 337 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 353 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 369 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 385 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 401 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 417 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 433 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 449 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 465 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 481 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |
| 497 |     |         | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 2 | 3 | 3 | 1 | 1 | 1 | 1 |

TABLE VI: Minus array for $L = 32$, $n = 512$ and $1 \leq k \leq n$. The value in the table is $\text{minus}_{512}(n, k+\Delta)$. 
| $n$  | $k$  | $\Delta$ |
|------|------|----------|
| 1    | 0    | 0  0  0 |
| 17   | 1    | 1  1  1 |
| 33   | 1    | 1  1  1 |
| 49   | 1    | 2  3  3 |
| 65   | 1    | 4  4  5 |
| 81   | 1    | 4  4  5 |
| 97   | 5    | 6  6  6 |
| 113  | 10   | 8  9  9 |
| 129  | 11   | 11 10 10 |
| 145  | 14   | 12 12 12 |
| 161  | 18   | 16 16 16 |
| 177  | 18   | 18 18 18 |
| 193  | 20   | 20 21 21 |
| 209  | 22   | 22 22 22 |
| 225  | 24   | 24 24 24 |
| 241  | 32   | 32 32 32 |
| 257  | 24   | 24 24 24 |
| 273  | 42   | 42 42 42 |
| 289  | 48   | 48 48 48 |
| 305  | 54   | 54 54 54 |
| 321  | 60   | 60 60 60 |
| 337  | 65   | 65 65 65 |
| 353  | 72   | 72 72 72 |
| 369  | 80   | 80 80 80 |
| 385  | 29   | 29 29 29 |
| 401  | 32   | 32 32 32 |
| 417  | 93   | 93 93 93 |
| 433  | 100 | 100 100 100 |
| 449  | 108 | 111 111 111 |
| 465  | 122 | 119 120 120 |
| 481  | 137 | 128 128 128 |
| 497  | 151 | 141 142 142 |
| 513  | 154 | 155 154 154 |
| 529  | 160 | 161 162 162 |
| 545  | 163 | 164 164 164 |
| 561  | 166 | 166 166 166 |
| 577  | 169 | 169 169 169 |
| 593  | 172 | 173 175 175 |
| 609  | 176 | 177 176 176 |
| 625  | 180 | 183 183 183 |
| 641  | 184 | 185 185 185 |
| 657  | 188 | 189 189 189 |
| 673  | 192 | 192 192 192 |
| 689  | 196 | 196 196 196 |
| 705  | 200 | 201 201 201 |
| 721  | 204 | 205 205 205 |
| 737  | 208 | 209 209 209 |
| 753  | 212 | 213 213 213 |
| 769  | 216 | 217 217 217 |
| 785  | 220 | 221 221 221 |
| 801  | 224 | 225 225 225 |
| 817  | 228 | 229 229 229 |
| 833  | 232 | 233 233 233 |
| 849  | 236 | 237 237 237 |
| 865  | 240 | 241 241 241 |
| 881  | 244 | 245 245 245 |
| 897  | 248 | 249 249 249 |
| 913  | 252 | 253 253 253 |
| 929  | 256 | 257 257 257 |
| 945  | 260 | 261 261 261 |
| 961  | 264 | 265 265 265 |
| 977  | 268 | 269 269 269 |
| 993  | 272 | 273 273 273 |
| 1009 | 276 | 277 277 277 |

**TABLE VII:** Minus array for $L = 32$, $n = 1024$ and $1 \leq k \leq n$. The value in the table is $\text{minus}_{32}(n, k + \Delta)$. 
REFERENCES

[1] 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.

[2] I. Tal and A. Vardy, “How to construct polar codes,” *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6562–6582, 2013.

[3] R. Mori and T. Tanaka, “Performance and construction of polar codes on symmetric binary-input memoryless channels,” in *2009 IEEE International Symposium on Information Theory*, 2009, pp. 1496–1500.

[4] ——, “Performance of polar codes with the construction using density evolution,” *IEEE Communications Letters*, vol. 13, no. 7, pp. 519–521, 2009.

[5] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[6] K. Niu and K. Chen, “CRC-aided decoding of polar codes,” *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, 2012.

[7] 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.

[8] 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.

[9] E. Abbe, A. Shpilka, and M. Ye, “Reed–Muller codes: Theory and algorithms,” *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3251–3277, 2021.

[10] 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.

[11] 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.