A Deterministic Algorithm for Computing the Weight Distribution of Polar Code

Hanwen Yao, *Member, IEEE,* Arman Fazeli, *Member, IEEE,* and Alexander Vardy, *Fellow, IEEE*

***Abstract*—In this work, we present a deterministic algorithm for computing the entire weight distribution of polar codes. As the first step, we derive an efficient recursive procedure to compute the weight distribution that arises in successive cancellation decoding of polar codes along any decoding path. This solves the open problem recently posed by Polyanskaya, Davletshin, and Polyanskii. Using this recursive procedure, at code length n, we can compute the weight distribution of any *polar cosets* in time $O(n^2)$. We show that any polar code can be represented as a disjoint union of such polar cosets; moreover, this representation extends to polar codes with dynamically frozen bits. However, the number of polar cosets in such representation scales exponentially with a parameter introduced herein, which we call the *mixing factor*. To upper bound the complexity of our algorithm for polar codes being decreasing monomial codes, we study the range of their mixing factors. We prove that among all decreasing monomial codes with rates at most 1/2, self-dual Reed-Muller codes have the largest mixing factors. To further reduce the complexity of our algorithm, we make use of the fact that, as decreasing monomial codes, polar codes have a large automorphism group. That automorphism group includes the block lower-triangular affine group (BLTA), which in turn contains the lower-triangular affine group (LTA). We prove that a subgroup of LTA acts transitively on certain subsets of decreasing monomial codes, thereby drastically reducing the number of polar cosets that we need to evaluate. This complexity reduction makes it possible to compute the weight distribution of polar codes at length $n = 128$.**

***Index Terms*—Polar codes, decreasing monomial codes, weight distribution**

I. Introduction

The weight distribution of an error correction code counts the number of codewords in this code of all weights. The weight distribution is one of the main characteristic of a code, useful for analysing its performance under maximum-likelihood decoding, and various other decoding algorithms. However, computing the weight distribution of a general linear code is known to be NP-hard [1]. Hence, there are very few families of codes whose weight distribution is known. Some families of codes with known weight distributions are Hamming codes, Golay codes, and Reed-Solomon codes. For primitive Bose-Chaudhuri-Hocquenghem (BCH) codes and the extended primitive BCH codes, their weight distributions are known for lengths up to 128 [2]. Besides, the weight distributions for primitive BCH codes of length 255 and extended primitive BCH codes of length 256 have been computed for code dimension $k \leq 71$ and $k \geq 187$ [3]–[5]. For Reed-Muller codes, their weight distributions are known up to length 512, except for the (512,256) Reed-Muller code [6], [7]. Polar codes, introduced by Arıkan [8], form the first explicit family of codes that provably achieve capacity with efficient encoding and decoding for a wide range of channels. With the brute-force search and the MacWilliams identity [9], the weight distribution of polar codes can be computed up to length 64. For example, the weight distribution of a (64,32) polar code is computed by brute force in [10] for code design purpose. However, the weight distribution of polar codes at length 128 is currently not known.

The work of Hanwen Yao, Arman Fazeli and Alexander Vardy was supported in part by the United States National Science Foundation (NSF) under Grants CCF-1405119 and Grant CCF-1719139. An earlier version of this paper was presented in part at the 2021 International Symposium on Information Theory (ISIT). (Corresponding author: Hanwen Yao.)

Hanwen Yao is with Duke University, Durham, NC 27708 USA (e-mail: hanwen.yao@duke.edu).

Arman Fazeli is with Apple Inc., Los Altos, CA 94022 USA (e-mail: afazelic@ucsd.edu).

Alexander Vardy is with the University of California at San Diego, San Diego, CA 92093 USA (e-mail: vardy@ece.ucsd.edu).

A. Related Prior Work

Introduced by Arıkan [8], polar code is a binary linear code generated by a subset of rows in the polar transformation matrix, whose corresponding bit channels have the smallest Bhattacharyya parameters. To the best of our knowledge, there are no prior results on how to efficiently compute the entire weight distribution of polar codes. For crude estimations, there are probabilistic methods discussed in [11] and [12]. Although we don't know the weight distribution of polar codes, we do know their minimal weight, and the number of codewords of that weight. In the work by Bardet, Dragoi, Otmani and Tillich [13], they look into a partial order relation for the bit channels, and introduce a broader class of codes following the partial order called decreasing monomial codes. This class of decreasing monomial codes includes polar codes. They also study the automorphism group of decreasing monomial codes from a polynomial formalism, and provide an explicit formula for the number of codewords of minimal weight. Besides that result, there are also ways to estimate the first few numbers in the weight distribution of polar codes. In the work by Li, Shen and Tse [14], they devise an experiment that evaluates the number of low-weight polar codewords. In this experiment, an all-zero codeword is transmitted in the extremely high SNR regime, and the channel output is decoded by a list decoder [15]. With a large enough list size, the first few non-zero numbers in the weight distribution can be estimated by counting the low-weight codewords obtained in the list. Later in [16], this experiment is improved for a memory constraint computer. But still, with this approach, only the first few numbers in the entire weight distribution can be estimated. This approach is also *non-exact* in the sense that, for a given weight, the number of codewords obtained in the list only serves as a lower bound for the actual number in the weight distribution.

B. Our Contributions

In this paper, we present a deterministic algorithm that computes the *exact* weight distribution of polar codes. We first propose an efficient recursive procedure to compute the weight enumerating function of *polar cosets* to be defined later. Those polar cosets arise during the successive cancellation (SC) decoding process, and their weight distribution can be used to estimate the error probabilities of the bit channels. In two separate works by Niu, Li, and Wu [17], and by Polyanskaya, Davletshin, and Polyanskii [18], algorithms that compute the weight distribution of these polar cosets along the all-zero decoding path are proposed. However, how to efficiently compute the weight distribution of polar cosets along an arbitrary decoding path remains an open problem. In this work, we solve this problem by establishing a recursive relation followed by the weight enumerating functions of those polar cosets. Using this recursive relation, we can compute the weight distribution of polar cosets along arbitrary decoding path in time $O(n^2)$.

Next, we show that we can represent any polar code as a disjoint union of certain polar cosets. In this way, we can obtain the weight distribution of the entire code by summing up the weight enumerating functions of those polar cosets. This representation also extends to polar codes with dynamically frozen bits, which are first introduced in [19]. Since any binary linear codes can be represented as polar codes with dynamically frozen bits [19], our algorithm applies to general linear codes as well. However, the number of polar cosets in this representation scales exponentially with a code parameter introduced herein, which we call the mixing factor. The complexity of our algorithm is largely governed by the mixing factor of polar codes.

Representing polar codes as disjoint unions of polar cosets works for polar codes in a general setting, where we can select any subsets of rows in the polar transformation matrix as generators. In a more restricted setting, where we only select the rows whose corresponding bit channels have the smallest Bhattacharyya parameters, polar codes fall into the category of decreasing monomial codes [13]. To upper bound the mixing factor of polar codes being decreasing monomial codes, and thus give a bound on the complexity of our algorithm, we prove that self-dual Reed-Muller codes have the largest mixing factor among all decreasing monomial codes with rates at most 1/2.

As decreasing monomial codes, polar codes have a large automorphism group. It is first shown in [13] that the automophism group of decreasing monomial codes includes the lower triangular affine group (LTA). Recently in [20], this result has been extended to the block lower triangular affine group (BLTA). Later in [21], it has also been shown that BLTA equals the complete automorphisms of decreasing monomial codes that can be formulated as affine transformations. In our work, we show that using a subgroup of LTA, we can largely reduce the complexity of our algorithm. We prove that the subgroup we considered acts transitively on certain subsets of decreasing monomial codes, which implies that a lot of polar cosets in our representation share the same weight distribution. It allows us to drastically reduce the number of polar cosets that we need to evaluate in our algorithm. This complexity reduction makes it possible to compute the weight distribution of polar codes as a decreasing monomial codes at length 128.

C. Notations

Here we specify some notation conventions we follow in this paper. All the vectors in this paper are row vectors, unless otherwise specified. We use bold letters like $\boldsymbol{u}$ to denote vectors, and non-bold letters like u_i to denote symbols within that vector. We let the indices for the symbols within vectors start from zero. We use $\boldsymbol{u}_i$ to represent $(u_0, u_1, \cdots, u_i)$, a subvector of $\boldsymbol{u}$ with its first $(i+1)$ symbols. We denote the concatenation of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ as $(\boldsymbol{u}, \boldsymbol{v})$.

II. Polar Codes and Polar Cosets

In this section, we briefly review polar codes, and give the definition for polar cosets, an essential concept in our work.

Assuming $n = 2^m$, an (n, k) polar code is a binary linear block code generated by k rows in the polar transformation matrix $G_n = B_n K_2^{\otimes m}$, where B_n is the bit-reversal permutation matrix, $K_2^{\otimes m}$ is the m-th Kronecker power of K_2, and

$$K_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

The encoding of polar codes is given by $\boldsymbol{c} = \boldsymbol{u}G_n$, where $\boldsymbol{u}$ is a length-n binary input vector carrying k data bits, and $\boldsymbol{c}$ is the codeword for transmission. The positions of the k data bits in $\boldsymbol{u}$ are specified by an information index set $\mathcal{A}$ of size k, with $\mathcal{A} \subseteq \{0, 1, \cdots, n-1\}$. The remaining $n-k$ bits in $\boldsymbol{u}$ are set to 0, which are called frozen bits. We also use $\mathcal{F} = \{0, 1, \cdots, n-1\} \backslash \mathcal{A}$ to denote the frozen index set that specifies the positions of the frozen bits.

We now give the definition for polar cosets.

Definition 1: For a vector $\boldsymbol{u}_i \in \{0,1\}^{i+1}$ with $0 \le i \le n-1$, we define the *polar coset* for path $\boldsymbol{u}_i$ as the affine space

$$C_n(\boldsymbol{u}_i) \triangleq \left\{(\boldsymbol{u}_i, \boldsymbol{u}')G_n \mid \boldsymbol{u}' \in \{0,1\}^{n-i-1}\right\}$$

where $(\boldsymbol{u}_i, \boldsymbol{u}')$ represents the concatenation of $\boldsymbol{u}$ and $\boldsymbol{u}'$, and G_n is the polar transformation matrix.

Example 1: Consider polar transformation matrix G_8 with its rows denoted by $g_0, g_1, \cdots, g_7$ as shown in Figure 1.

Let $\boldsymbol{u}_4 = (0, 1, 0, 1, 0)$, then the polar coset $C_8(\boldsymbol{u}_4)$ is the affine space generated by g_5, g_6, g_7, and shifted by g_1 and g_3:

$$C_8(\boldsymbol{u}_4) = g_1 + g_3 + \operatorname{span}\{g_5, g_6, g_7\}$$

In Figure 1, those rows are highlighted in gray and in cyan, respectively.

In this paper, we will mainly discuss the weight distribution of polar cosets, which can be described by their weight enumerating functions.

$$G_8 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix} \begin{matrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \end{matrix}$$

Fig. 1. Polar transformation matrix G_8 in Example 1

Definition 2: For a vector $\boldsymbol{u}_i \in \{0,1\}^{i+1}$ with $0 \leq i \leq n-1$, we define the weight enumerating function for polar coset $C_n(\boldsymbol{u}_i)$ as the polynomial

$$A_n(\boldsymbol{u}_i)(X) \triangleq \sum_{w=0}^{n} A_w X^w,$$

where A_w is the number of vectors in $C_n(\boldsymbol{u}_i)$ with Hamming weight w.

In prior works, the weight distribution of polar coset $C_n(\boldsymbol{u}_i)$, where $\boldsymbol{u}_i = (0,0,\cdots,0,1)$ is a length-$(i+1)$ all-zero vector with a single 1 at the end, is also referred to as *polar spectrum* in [17], and as the *weight distribution for SC decoding of polar codes* in [18]. It has also been pointed out in [17, Sec.III.B] and [18, Sec.II.C] that the weight distribution of such polar coset $C_n(\boldsymbol{u}_i)$ can be used to analyze the error probability of the bit channels.

III. Computing the Weight Enumerating Function of Polar Cosets

In this section, we present the first key result of this paper: a recursive procedure that computes the weight enumerating function for arbitrary polar cosets. Recently, the authors in [17] and [18] have introduced their respective algorithms that compute the weight distribution for polar coset $C_n(\boldsymbol{u}_i)$ with $\boldsymbol{u}_i = (0,0,\cdots,0,0)$ and $\boldsymbol{u}_i = (0,0,\cdots,0,1)$. However, how to efficiently compute the weight distribution for $C_n(\boldsymbol{u}_i)$ with arbitrary path $\boldsymbol{u}_i$ remains an open problem. Here we present a recursive computation procedure with time complexity $O(n^2)$ that solves this problem.

Let us first establish some notations. We use $\boldsymbol{u}_{\text{even}}$ and $\boldsymbol{u}_{\text{odd}}$ to denote the subvectors $(u_0, u_2, \cdots)$ and $(u_1, u_3, \cdots)$ of $\boldsymbol{u}$ with only even indices and only odd indices, respectively. We use $\boldsymbol{u}_{i,\text{even}}$ and $\boldsymbol{u}_{i,\text{odd}}$ to denote the subvectors of $\boldsymbol{u}_i$ with only even indices and only odd indices, respectively.

Our algorithm for polar cosets is based on the following recursive relations.

Theorem 1: Let $m \geq 0$, $n = 2^m$, and $0 \leq i \leq n-1$, then

$$A_{2n}(\boldsymbol{u}_{2i})(X) = \sum_{u_{2i+1}\in\{0,1\}} A_n(\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}))(X) \cdot A_n(\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1})(X), \quad (1)$$

and

$$A_{2n}(\boldsymbol{u}_{2i+1})(X) = A_n(\boldsymbol{u}_{2i+1,\text{even}} \oplus \boldsymbol{u}_{2i+1,\text{odd}})(X) \cdot A_n(\boldsymbol{u}_{2i+1,\text{odd}})(X). \quad (2)$$

Proof: Let $m \geq 0$ and $n = 2^m$. For any $\boldsymbol{u} \in \{0,1\}^{2n}$, we have

$$\begin{aligned} \boldsymbol{u} \cdot G_{2n} &= (\boldsymbol{u} \cdot B_{2n}) K_2^{\otimes(m+1)} \\ &= (\boldsymbol{u}_{\text{even}} \cdot B_n,\, \boldsymbol{u}_{\text{odd}} \cdot B_n) \begin{bmatrix} K_2^{\otimes m} & 0 \\ K_2^{\otimes m} & K_2^{\otimes m} \end{bmatrix} \\ &= \Big((\boldsymbol{u}_{\text{even}} \oplus \boldsymbol{u}_{\text{odd}}) \cdot B_n K_2^{\otimes m},\, \boldsymbol{u}_{\text{odd}} \cdot B_n K_2^{\otimes m} \Big) \\ &= \Big((\boldsymbol{u}_{\text{even}} \oplus \boldsymbol{u}_{\text{odd}}) \cdot G_n,\, \boldsymbol{u}_{\text{odd}} \cdot G_n \Big) \end{aligned} \quad (3)$$

We first prove equation (1). According to Definition 1, we have

$$C_{2n}(\boldsymbol{u}_{2i}) = \left\{ (\boldsymbol{u}_{2i}, \boldsymbol{u}') G_{2n} \mid \boldsymbol{u}' \in \{0,1\}^{2n-2i-1} \right\} \quad (4)$$

Let us represent $\boldsymbol{u}'$ as $\boldsymbol{u}' = (u_{2i+1}, \boldsymbol{v})$. By looking at the two values u_{2i+1} can take, $C_{2n}(\boldsymbol{u}_{2i})$ in (4) can be partitioned as:

$$\begin{aligned} &C_{2n}(\boldsymbol{u}_{2i}) \\ &= \bigcup_{u_{2i+1}\in\{0,1\}} C_{2n}(\boldsymbol{u}_{2i}, u_{2i+1}) \\ &= \bigcup_{u_{2i+1}\in\{0,1\}} \left\{ (\boldsymbol{u}_{2i}, u_{2i+1}, \boldsymbol{v}) G_{2n} \mid \boldsymbol{v} \in \{0,1\}^{2n-2i-2} \right\} \end{aligned} \quad (5)$$

Via (3), we can write $(\boldsymbol{u}_{2i}, u_{2i+1}, \boldsymbol{v}) G_{2n}$ in (5) as

$$\begin{aligned} &(\boldsymbol{u}_{2i}, u_{2i+1}, \boldsymbol{v}) G_{2n} = \\ &\Big((\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}),\, \boldsymbol{v}_{\text{even}} \oplus \boldsymbol{v}_{\text{odd}}) \cdot G_{2n}, \\ &\qquad (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}, \boldsymbol{v}_{\text{odd}}) \cdot G_{2n} \Big) \end{aligned} \quad (6)$$

Notice when $\boldsymbol{v}$ ranges over $\{0,1\}^{2n-2i-2}$, both $\boldsymbol{v}_{\text{odd}}$ and $(\boldsymbol{v}_{\text{even}} \oplus \boldsymbol{v}_{\text{odd}})$ range over $\{0,1\}^{n-i-1}$ separately. Thus we have

$$\begin{aligned} &C_{2n}(\boldsymbol{u}_{2i}, u_{2i+1}) = \\ &\Big\{ (\boldsymbol{c}_1, \boldsymbol{c}_2) \mid \boldsymbol{c}_1 \in C_n(\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1})), \\ &\qquad \boldsymbol{c}_2 \in C_n(\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}) \Big\} \end{aligned} \quad (7)$$

Hence for each $u_{2i+1} \in \{0,1\}$, $C_{2n}(\boldsymbol{u}_{2i}, u_{i+1})$ in (5) can be expressed as the *direct sum* of two polar cosets [22, §9 of Ch. 2]. In other words, $C_{2n}(\boldsymbol{u}_{2i}, u_{i+1})$ consists of all vectors $(\boldsymbol{c}_1, \boldsymbol{c}_2)$, where $\boldsymbol{c}_1 \in C_n(\boldsymbol{u}_{2i,\text{even}} \oplus (\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1}))$, and $\boldsymbol{c}_2 \in C_n(\boldsymbol{u}_{2i,\text{odd}}, u_{2i+1})$.

Since the weight enumerating function of the direct sum of two polar cosets equals the product of their two individual weight distribution functions, we obtain equation (1). Equation (2) follows in the same way by rewritting (7) as

$$\begin{aligned} &C_{2n}(\boldsymbol{u}_{2i+1}) = \\ &\Big\{ (\boldsymbol{c}_1, \boldsymbol{c}_2) \mid \boldsymbol{c}_1 \in C_n(\boldsymbol{u}_{2i+1,\text{even}} \oplus \boldsymbol{u}_{2i+1,\text{odd}}), \\ &\qquad \boldsymbol{c}_2 \in C_n(\boldsymbol{u}_{2i+1,\text{odd}}) \Big\} \end{aligned} \quad (8)$$

■

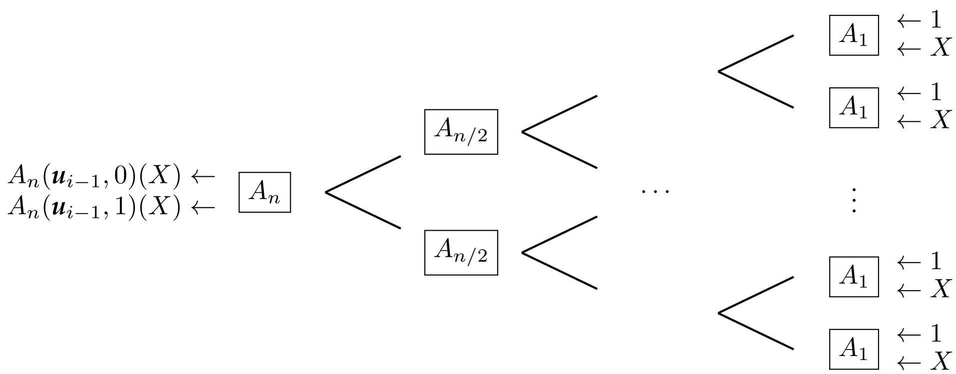

Fig. 2. The recursive procedure that computes the weight enumerating function for polar cosets

In Theorem 1, equation (1) and equation (2) can also be written as

$$A_{2n}(\boldsymbol{u}_{2i-1},u_{2i})(X) = \sum_{u_{2i+1}\in\{0,1\}} A_n(\boldsymbol{u}_{2i-1,\text{even}} \oplus \boldsymbol{u}_{2i-1,\text{odd}}\,,\, u_{2i}\oplus u_{2i+1})(X) \cdot A_n(\boldsymbol{u}_{2i-1,\text{odd}}, u_{2i+1})(X) \quad (9)$$

and as

$$A_{2n}(\boldsymbol{u}_{2i},u_{2i+1})(X) = A_n(\boldsymbol{u}_{2i-1,\text{even}} \oplus \boldsymbol{u}_{2i-1,\text{odd}},\, u_{2i}\oplus u_{2i+1})(X) \cdot A_n(\boldsymbol{u}_{2i-1,\text{odd}}, u_{2i+1})(X), \quad (10)$$

respectively. In this way, equation (9) and equation (10) fall into forms similar to the recursive relations for the bit channels [8, Equations (22) and (23)]. Therefore, similar to the recursive procedure that computes the probabilities for the bit channels, we can also compute the weight enumerating functions of polar cosets recursively with the stopping conditions:

$$A_1(0) = 1, \quad A_1(1) = X. \quad (11)$$

This recursive procedure is illustrated in Figure 2, and its steps are shown in Algorithm 1.

We make the following remarks for Algorithm 1:

- The object for recursion in Algorithm 1 is a pair of weight enumerating functions $A_n(\boldsymbol{u}_{i-1},0)(X)$ and $A_n(\boldsymbol{u}_{i-1},1)(X)$.
- If we want to compute the weight distribution for polar coset $C_n(\boldsymbol{u}_i)$, we should run Algorithm 1 with inputs n and $\boldsymbol{u}_{i-1}$, and select one of the two weight enumerating functions from the output corresponding to the desired u_i.

Next, we prove that Algorithm 1 has time complexity $O(n^2)$.

Theorem 2: Algorithm 1 has time complexity $O(n^2)$.

Proof: In Algorithm 1, depending on the inputs i and $\boldsymbol{u}_{i-1}$, we have the following three cases for the lines we need to run:

Case 1: When i is even, we run lines 5, 6 and 7.
Case 2: When i is odd and $u_i = 0$, we run lines 9, 10 and 12.
Case 3: When i is odd and $u_i = 1$, we run lines 9, 10 and 14.

Algorithm 1: CalcA$(n, \boldsymbol{u}_{i-1})$

Input: block length n and vector $\boldsymbol{u}_{i-1}$
Output: a pair of polynomials $(A_n(\boldsymbol{u}_{i-1},0)(X), A_n(\boldsymbol{u}_{i-1},1)(X))$

1 **if** $n = 1$ **then** // Stopping conditions
2   **return** $(1, X)$
3 **else**
4   **if** $i \mod 2 = 0$ **then**
5     $(f_0, f_1) \leftarrow$ CalcA$(n/2, \boldsymbol{u}_{i-1,\text{even}} \oplus \boldsymbol{u}_{i-1,\text{odd}})$
6     $(g_0, g_1) \leftarrow$ CalcA$(n/2, \boldsymbol{u}_{i-1,\text{odd}})$
7   **return** $(f_0g_0 + f_1g_1, f_0g_1 + f_1g_0)$
8   // Use (1)
9   **else**
10     $(f_0, f_1) \leftarrow$ CalcA$(n/2, \boldsymbol{u}_{i-2,\text{even}} \oplus \boldsymbol{u}_{i-2,\text{odd}})$
11     $(g_0, g_1) \leftarrow$ CalcA$(n/2, \boldsymbol{u}_{i-2,\text{odd}})$
12   **if** $u_{i-1} = 0$ **then**
13   **return** (f_0g_0, f_1g_1) // Use (2)
14   **else**
15   **return** (f_1g_0, f_0g_1) // Use (2)

First, the complexity of line 5 is the same as that of line 9, and the complexity of line 6 is the same as that of line 10. Then for line 7, we need to do 4 polynomial multiplications and 2 polynomial additions, while for line 12 or line 14, we only need to do 2 polynomial multiplications. So case 1 has the highest complexity among the above three cases. Thus, henceforth we only consider case 1.

Denote by $T(n)$ the time complexity of Algorithm 1. For the recursive part in the algorithm, line 5 and line 6 both take time $T(n/2)$. For the non-recursive part, in line 7 we need to do 4 polynomial multiplications and 2 polynomial additions. Since f_0, f_1, g_0, g_1 are weight enumerating functions of polar cosets with block length $n/2$, all of them have degrees at most $n/2$. Assume multiplication of two degree-n polynomials takes time $O(n^2)$, and addition of two degree-n polynomials takes time $O(n)$. It follows that

$$T(n) \leq 2T(n/2) + 4 \cdot O(n^2/4) + 2 \cdot O(n),$$

which by the Master theorem [23] gives us $T(n) = O(n^2)$. ■

We also remark that the time complexity of Algorithm 1 may be improved assuming multiplication of two degree-n polynomials takes time $O(n \log n)$ with the Fast-Fourier Transform.

IV. Computing the Entire Weight Distribution of Polar Codes

In this section, we present a deterministic algorithm that computes the entire weight distribution of polar codes. We first show that any polar code can be represented as a disjoint union of certain polar cosets. This allows us to obtain the weight distribution of the entire code by summing up the weight distributions of those polar cosets. However, the number of polar cosets in this representation scales exponentially with a new parameter that we introduce herein, called the mixing factor. We also show that our approach naturally extends to polar codes with dynamically frozen bits.

A. Representing Polar Codes with Polar Cosets

First, we introduce two new parameters of polar codes that we call the last frozen index and the mixing factor, respectively.

Definition 3: Consider an (n, k) polar code $\mathbb{C}$ specified in terms of its information index set $\mathcal{A}$. With $\mathcal{F} = \{0, 1, \ldots, n-1\}\backslash\mathcal{A}$, we define the *last frozen index* of $\mathbb{C}$ as

$$\tau(\mathbb{C}) \triangleq \max\{\mathcal{F}\},$$

and define the *mixing factor* of $\mathbb{C}$ as

$$\mathrm{MF}(\mathbb{C}) \triangleq \big|\{i \in \mathcal{A} \mid i < \tau(\mathbb{C})\}\big|.$$

Loosely speaking, the mixing factor of $\mathbb{C}$ counts the number of information bits appear before the last frozen bit. It is easy to see that $\mathrm{MF}(\mathbb{C})$ can be computed from $\tau(\mathbb{C})$ as follows:

$$\begin{aligned} \mathrm{MF}(\mathbb{C}) &= k - \big|\{i \in \mathcal{A} \mid i > \tau(\mathbb{C})\}\big| \\ &= \tau(\mathbb{C}) - (n-k) + 1 \end{aligned} \tag{12}$$

Starting with an example, we now show that any polar code can be represented as a disjoint union of polar cosets.

Example 2: In this example, we denote the (16,11,4) extended Hamming code as $\mathbb{C}_H$. It can be generated by rows in the polar transformation matrix G_{16}. Thus we can view $\mathbb{C}_H$ as a polar code of length 16 with frozen index set $\mathcal{F} = \{0, 1, 2, 4, 8\}$. The polar transformation matrix G_{16} is shown in Figure 3.

$$\begin{array}{c} u_0\\ u_1\\ u_2\\ u_3\\ u_4\\ u_5\\ u_6\\ u_7\\ u_8\\ u_9\\ u_{10}\\ u_{11}\\ u_{12}\\ u_{13}\\ u_{14}\\ u_{15} \end{array} \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\ 1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\ 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\ 1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\ 1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\ 1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\ 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\ 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{bmatrix}$$

Fig. 3. Polar transformation matrix G_{16} in Example 2

In Figure 3, the information bits of $\mathbb{C}_H$ are highlighted in red and blue, and the frozen bits are black. We color the information bits appearing before the last frozen bit in red, and color the rest of the information bits in blue. The last frozen bit of $\mathbb{C}_H$ is u_8, so the last frozen index of $\mathbb{C}_H$ is $\tau(\mathbb{C}_H) = 8$. The mixing factor of $\mathbb{C}_H$ counts the number of red bits, so the mixing factor of $\mathbb{C}_H$ is $\mathrm{MF}(\mathbb{C}_H) = 4$.

Consider the polar coset $C_{16}(\boldsymbol{u}_8)$. For any binary vector $\boldsymbol{u}_8$ with $u_0 = u_1 = u_2 = u_4 = u_8 = 0$, and $u_3, u_5, u_6, u_7 \in \{0, 1\}$, the polar coset $C_{16}(\boldsymbol{u}_8)$ will be a subset of $\mathbb{C}_H$. In total we have $2^4 = 16$ options to assign the values for u_3, u_5, u_6, u_7. Hence there are 16 such disjoint polar cosets, and the union of them is the entire code $\mathbb{C}_H$:

$$\mathbb{C}_H = \bigcup_{\boldsymbol{u}_8 \in \{0,1\}^9:\ u_0 = u_1 = u_2 = u_4 = u_8 = 0} C_{16}(\boldsymbol{u}_8)$$

Therefore, the entire weight distribution of $\mathbb{C}_H$ can be obtained by first computing the weight enumerating functions of all those 16 polar cosets, and then taking the sum.

This polar coset representation for general polar codes can be summarized by the following proposition.

Proposition 1: Let $\mathbb{C}$ be a polar code with frozen index set $\mathcal{F}$, and last frozen index τ. Then $\mathbb{C}$ can be represented as a disjoint union of polar cosets as:

$$\mathbb{C} = \bigcup_{\boldsymbol{u}_\tau \in \{0,1\}^{\tau+1}:\ u_i = 0 \text{ for all } i \in \mathcal{F}} C_n(\boldsymbol{u}_\tau)$$

The number of polar cosets in this representation equals $2^{\mathrm{MF}(\mathbb{C})}$.

B. Representing Polar Codes with Dynamically Frozen Bits

We now show that our polar coset representation in Proposition 1 extends to polar codes with *dynamically frozen bits*. Polar codes with dynamically frozen bits, first introduced in [19], are polar codes where each of the frozen bits u_i is not necessarily fixed to be zero, but can be set as a linear function of its previous bits as $u_i = f_i(\boldsymbol{u}_{i-1})$. For frozen bits with indices in $\mathcal{F}$, we refer to those boolean functions $\{f_i \mid i \in \mathcal{F}\}$ as the *dynamic constraints* for the code. Examples of polar codes with dynamically frozen bits are polar codes with CRC precoding [15], polar subcodes [24], polarization-adjusted convolutional (PAC) codes [25], etc. In fact, since any binary linear code can be represented as a polar code with dynamically frozen bits [19], our representation extends to all binary linear codes, as well.

The concept of last frozen index and mixing factor in Definition 3 naturally extends to polar codes with dynamically frozen bits. We again illustrate our polar coset representation with an example, in which the Hamming code in Example 2 is slightly modified so its frozen bits become dynamically frozen.

Example 3: Denote by $\mathbb{C}'_H$ a $(16, 11)$ polar code with frozen index set $\mathcal{F} = \{0, 1, 2, 4, 8\}$, where u_0, u_1, u_2 are frozen as 0, and u_4 and u_8 are dynamically frozen as $u_4 = u_3$ and $u_8 =$

$$
\begin{array}{l}
u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 = u_3 \\ u_5 \\ u_6 \\ u_7 \\ u_8 = u_5 + u_6 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \\ u_{15}
\end{array}
\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}
$$

Fig. 4. Polar transformation matrix G_{16} in Example 3

$u_5 + u_6$, respectively. We have $\tau(\mathbb{C}'_H) = 8$ and $\mathrm{MF}(\mathbb{C}'_H) = 4$ the same as in Example 2.

Consider the polar coset $C_{16}(\boldsymbol{u}_8)$. For any binary vector $\boldsymbol{u}_8$ with $u_3, u_5, u_6, u_7 \in \{0,1\}$, if we let $u_0 = u_1 = u_2 = 0$, $u_4 = u_3$ and $u_8 = u_5 + u_6$, then the polar coset $C_{16}(\boldsymbol{u}_8)$ will be a subset of $\mathbb{C}'_H$. Thus similar to Example 2, with $2^4 = 16$ options to assign the values for u_3, u_5, u_6 and u_7, $\mathbb{C}'_H$ can be represented as a disjoint union of 16 disjoint polar cosets as

$$\mathbb{C} = \bigcup_{\substack{\boldsymbol{u}_8 \in \{0,1\}^9:\ u_0=u_1=u_2=0,\ u_4=u_3,\\ u_8=u_5+u_6}} C_{16}(\boldsymbol{u}_8)$$

In general, Proposition 1 extends to polar codes with dynamically frozen bits as follows.

Proposition 2: Let $\mathbb{C}$ be a polar code with dynamically frozen bits, with frozen index set $\mathcal{F}$, last frozen index τ, and the dynamic constraints $\{f_i \mid i \in \mathcal{F}\}$. Then $\mathbb{C}$ can be represented as a disjoint union of polar cosets as:

$$\mathbb{C} = \bigcup_{\boldsymbol{u}_\tau \in \{0,1\}^{\tau+1}:\ u_i = f_i(\boldsymbol{u}_{i-1}) \text{ for all } i \in \mathcal{F}} C_n(\boldsymbol{u}_\tau)$$

The number of polar cosets in this representation equals $2^{\mathrm{MF}(\mathbb{C})}$.

C. Computing the Entire Weight Distribution

This polar coset representation directly gives us a way to compute the weight distribution of polar codes. We can compute the weight enumerating function of each polar coset in the representation using Algorithm 1, and then take their sum to obtain the weight distribution of the entire code. This procedure is shown in Algorithm 2, in which conventional polar codes are considered as special cases of polar codes with dynamically frozen bits.

For a polar code $\mathbb{C}$, the number of polar cosets in the representation equals 2 to the power of $\mathrm{MF}(\mathbb{C})$. For each polar coset, both the computation of its dynamically frozen bits and the computation of its weight enumerating function via Algorithm 1 have complexity $O(n^2)$. Thus without parallel computation, Algorithm 2 has time complexity $O(2^{\mathrm{MF}(\mathbb{C})}\, n^2)$. It is clear that this complexity is largely governed by the mixing factor of the code. For reference, we list the mixing factors of several rate 1/2 polar codes from length 8 to length 1024 in Table I. Those polar codes are constructed using the reliability sequence in the 5G technical specification [26].

Algorithm 2: Compute the weight enumerating functions of polar codes with dynamically frozen bits

Input: block length n, frozen index set $\mathcal{F}$, and dynamic constraint $\{f_i \mid i \in \mathcal{F}\}$
Output: weight enumerating function $A_{\mathbb{C}}(X)$ of polar code $\mathbb{C}$

```
τ ← max{F}
A_C(X) ← 0
for u_τ ∈ {0,1}^{τ+1} : u_i = f_i(u_{i−1}) for all i ∈ F do
    // Use Algorithm 1
    (f_0, f_1) ← CalcA(n, u_{τ−1})
    u_τ ← f_τ(u_{τ−1})
    if u_τ = 0 then
        A_C(X) ← A_C(X) + f_0
    else
        A_C(X) ← A_C(X) + f_1
return A_C(X)
```

TABLE I
MIXING FACTOR OF RATE 1/2 POLAR CODES CONSTRUCTED USING THE RELIABILITY SEQUENCE IN 5G [26]

code length n	8	16	32	64	128	256	512	1024
MF($\mathbb{C}$)	1	2	9	17	34	73	161	385

Unfortunately, this approach turns out to be inefficient for polar codes with CRCs. For a polar code concatenated with a CRC outer code [15], since all the CRC parity bits are located at the end of the data vector, the mixing factor of the code would be the same as the code dimension. In this case, Algorithm 2 will have complexity higher than that of the brute-force search.

For PAC codes [25], their mixing factors are determined by the rate profiles. For PAC codes with polar rate profiles, their mixing factors will be the same as polar codes. For PAC codes with Reed-Muller rate profiles, which show better performance under sequential decoding and list decoding [25], [27], [28], their mixing factors will be the same as Reed-Muller codes. As will be shown in Section V, Reed-Muller codes have relatively larger mixing factors compared with polar codes.

We also list the mixing factors of several extended BCH codes represented as polar codes with dynamically frozen bits in Table II. Those codes are obtained by extending some of the primitive narrow-sense BCH codes listed in Table A-1 in [29, Appendix A]. Note that for a given binary linear code, its representations as polar codes with dynamically frozen bits will be different for different codeword bit orders. Since it is known that primitive BCH codes contain as subcodes punctured Reed-Muller code of the same designed distance [22, Ch. 13. §5. Theorem 11], we permute the bit positions of those extended BCH codes from the *cyclic order* to the *standard order* [30], such that heuristically, their representations as polar codes with dynamically frozen bits have smaller mixing factors. This standard order is also used in [24] to construct polar subcodes

TABLE II
MIXING FACTORS OF EXTENDED BCH (EBCH) CODES AS POLAR CODES WITH DYNAMICALLY FROZEN BITS. THE DISTANCES OF THE CODES ARE OBTAINED FROM [31] AND FROM [2].

EBCH(n, k)	distance d	mixing factor $\mathrm{MF}(\mathbb{C})$
EBCH$(8, 4)$ = RM$(1, 3)$	4	1
EBCH$(16, 7)$	6	4
EBCH$(32, 16)$ = RM$(2, 5)$	8	9
EBCH$(64, 30)$	14	23
EBCH$(64, 36)$	12	29
EBCH$(128, 57)$	24	50
EBCH$(128, 64)$	22	57

from extended BCH codes. In Table II, we also specify the extended BCH codes that are equivalent to Reed-Muller codes RM(r,m) of order r and length $n = 2^m$.

Note that the weight distribution of extended BCH codes at length 128 have already been computed by Desaki, Fujiwara and Kasami in [2]. Here, we only list those mixing factors as a reference. Compared with polar codes, we can see that the mixing factors for extended BCH codes that have large code distances are also much larger, which indicates that our approach is less applicable here.

V. Mixing Factor of Polar Codes

Note that the approach described in Section IV applies to polar codes in a general setting where: (1) the frozen bits can be dynamically frozen; (2) the information index set can be arbitrary. Hereforth, we focus on conventional polar codes where: (1) the frozen bits are all frozen to zero; (2) for code of dimension k, the information index set $\mathcal{A}$ is chosen such that the corresponding bit channels are the "best" k bit channels. In Arıkan's definition, the k bit channels with the smallest Bhattacharyya parameters are selected. Two alternative criteria for picking the best k bit channels are mutual information and error probabilities. If we follow either one of these three criteria, polar codes fall into the category of decreasing monomial codes, first introduced in [13].

In this section, we briefly review the definition of decreasing monomial codes. Then, to upper bound the complexity of Algorithm 2, we prove that self-dual Reed-Muller codes have the largest mixing factors among all decreasing monomial codes with rates at most $1/2$.

A. Decreasing Monomial Codes

We start by reviewing the definition of monomial codes. Let $n = 2^m$, and let the polynomial ring given by

$$\mathcal{R}_m = \mathbb{F}[x_0, x_1, \cdots, x_{m-1}] \\ /(x_0^2 - x_0, x_1^2 - x_1, \cdots, x_{m-1}^2 - x_{m-1}).$$

Each polynomial $p \in \mathcal{R}_m$ can be associated with a binary vector in $\mathbb{F}_2^n$ as the evaluation of p in all the binary entries $\boldsymbol{x} = (x_0, \cdots, x_{m-1}) \in \mathbb{F}_2^m$. In other words, polynomial p is associated with $\mathrm{ev}(p) = (p(\boldsymbol{x}))_{\boldsymbol{x} \in \mathbb{F}_2^m}$ where $\mathrm{ev} : \mathcal{R}_m \to \mathbb{F}_2^n$ is a homomorphism from the polynomials to the binary n-tuples. In this work, we specify the order of $\boldsymbol{x}$ in vector

f	$[f]$	$\llbracket f \rrbracket$	
$x_0x_1x_2x_3$	$(0,0,0,0)$	0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$x_1x_2x_3$	$(0,0,0,1)$	1	1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
$x_0x_2x_3$	$(0,0,1,0)$	2	1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
x_2x_3	$(0,0,1,1)$	3	1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
$x_0x_1x_3$	$(0,1,0,0)$	4	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
x_1x_3	$(0,1,0,1)$	5	1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0
x_0x_3	$(0,1,1,0)$	6	1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0
x_3	$(0,1,1,1)$	7	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
$x_0x_1x_2$	$(1,0,0,0)$	8	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
x_1x_2	$(1,0,0,1)$	9	1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0
x_0x_2	$(1,0,1,0)$	10	1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0
x_2	$(1,0,1,1)$	11	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
x_0x_1	$(1,1,0,0)$	12	1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
x_1	$(1,1,0,1)$	13	1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
x_0	$(1,1,1,0)$	14	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1	$(1,1,1,1)$	15	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Fig. 5. Polar transformation matrix G_{16} in Example 4

$(p(\boldsymbol{x}))_{\boldsymbol{x} \in \mathbb{F}_2^m}$ such that from left to right, the number $\sum_{i=0}^{m-1} z_i 2^i$ is in ascending order from 0 to $2^m - 1$, where the binary vector $(z_0, z_1, \cdots, z_{m-1})$ is defined by:

$$(z_0, z_1, \cdots, z_{m-1}) = (1 - x_{m-1}, 1 - x_{m-2}, \cdots, 1 - x_0)$$

Denote the set of all the monomials in $\mathcal{R}_m$ as

$$\mathcal{M}_m = \left\{ x_0^{b_0} x_1^{b_1} \cdots x_{m-1}^{b_{m-1}} \;\middle|\; (b_0, b_1, \cdots, b_{m-1}) \in \mathbb{F}_2^m \right\}.$$

The monomial codes can be defined as follows.

Definition 4: Let $n = 2^m$ and $\mathcal{I} \in \mathcal{M}_m$, the monomial code $\mathbb{C}(\mathcal{I})$ generated by $\mathcal{I}$ is the linear space

$$\mathbb{C}(\mathcal{I}) \triangleq \mathrm{span}\{\mathrm{ev}(f) \mid f \in \mathcal{I}\}.$$

Since every row in the polar transformation matrix G_n can be obtained as $\mathrm{ev}(f)$ with some $f \in \mathcal{M}_m$, polar codes can be viewed as monomial codes. For a monomial $f \in \mathcal{M}_m$ given by $f = x_{i_1} x_{i_2} \cdots x_{i_d}$, we write:

$$\begin{aligned} \deg f &= d, \\ \mathrm{ind}(f) &= \{i_1, i_2, \ldots, i_d\}, \\ [f] &= (a_{m-1}, a_{m-2}, \ldots, a_0) \in \{0,1\}^m, \\ \llbracket f \rrbracket &= \sum_{i=0}^{m-1} a_i 2^i = \sum_{i=0}^{m-1} (1 - b_i) 2^i, \end{aligned}$$

where the two binary vectors $(a_{m-1}, a_{m-2}, \ldots, a_0)$ and $(b_{m-1}, b_{m-2}, \ldots, b_0)$ are defined by:

$$f = x_0^{1-a_0} x_1^{1-a_1} \cdots x_{m-1}^{1-a_{m-1}} = x_0^{b_0} x_1^{b_1} \cdots x_{m-1}^{b_{m-1}}$$

Following this notation, if we label the rows in the polar transformation matrix G_n with indices from 0 to $n-1$, the evaluation $\mathrm{ev}(f)$ for a monomial $f \in \mathcal{M}_m$ has row index $\llbracket f \rrbracket$ in G_n, and $[f]$ contains the digits in the binary expansion of $\llbracket f \rrbracket$. When the underlying G_n is clear from the context, we simply refer $\llbracket f \rrbracket$ as the *row index* for f.

Example 4: Consider the polar transformation matrix G_{16}. The monomials in $\mathcal{M}_4$ whose evaluations are rows in G_{16} are shown in Figure 5.

Henceforth, whenever we write a monomial as $f = x_{i_1} x_{i_2} \cdots x_{i_d}$ we assume that $i_1 < i_2 < \cdots < i_d$, unless stated otherwise. A partial order on the monomials in $\mathcal{M}_m$ is introduced in [13] as follows:

Definition 5: If $f = x_{i_1}x_{i_2}\cdots x_{i_d}$ and $g = x_{j_1}x_{j_2}\cdots x_{j_d}$ are two monomials of the same degree d, we write $f \preccurlyeq g$ if

$$i_1 \le j_1, \quad i_2 \le j_2, \quad \cdots, \quad i_d \le j_d$$

If $\deg f < \deg g$, we write $f \preccurlyeq g$ if there exists a divisor g^* of g, such that g^* has the same degree as f and $f \preccurlyeq g^*$. If $f \preccurlyeq g$ and $f \ne g$, we write $f \prec g$.

It has been shown by Bardet, Dragoi, Otmani, and Tillich in [13], and by Schürch in [32] that polar codes satisfy the following property.

Theorem 3: Let $\mathbb{C}_n(\mathcal{A})$ be a polar code, specified in terms of its information index set $\mathcal{A}$. If $[\![g]\!] \in \mathcal{A}$ and $f \preccurlyeq g$, then also $[\![f]\!] \in \mathcal{A}$. Equivalently, if $[\![f]\!] \in \mathcal{F}$ and $f \preccurlyeq g$, then also $[\![g]\!] \in \mathcal{F}$.

Therefore, the authors in [13] call the family of all codes having this property as decreasing monomial codes.

Definition 6 (Decreasing monomial codes [13]): Decreasing monomial codes inclues all monomial codes that satisfy Theorem 3.

Besides polar codes, the family of decreasing monomial codes also includes Reed-Muller codes. A simple lemma about the partial order of two monomials, and their row indices that is easy to verify is the following:

Lemma 1: If $g \preccurlyeq f$, then $[\![g]\!] \ge [\![f]\!]$.

B. The Largest Mixing Factor of Polar Codes

Now we are ready to study the range of mixing factor of decreasing monomial codes. Since by the MacWilliams identity [9], one can easily obtain the weight distribution of a code from the weight distribution of its dual, if we want to compute the weight distribution of a given decreasing monomial code, we have the options of applying Algorithm 2 to either the code itself, or to its dual. On the other hand, Bardet, Dragoi, Otmani, and Tillich have shown that the dual of any decreasing monomial code is also a decreasing monomial code [13, Proposition 6]. Thus to get a complexity cap of our approach, it suffices to limit our space to decreasing monomial codes of rates at most 1/2.

Theorem 4: Let $\mathbb{C}$ be an (n, k) decreasing monomial code with $n = 2^m$, $m = 2t + 1$, and dimension $k \le n/2$, then

$$\mathrm{MF}(\mathbb{C}) \le 2^{2t} - 2^{t+1} + 1 \tag{13}$$

Moreover, the equality holds only when $\mathbb{C}$ is the self-dual Reed-Muller code.

According to Theorem 4, the mixing factor of decreasing monomial codes at length $n = 2^m$, where m is an odd number, is bounded by the mixing factor of self-dual Reed-Muller codes. Here we list the mixing factor of self-dual Reed-Muller codes at length 8, 32, 128, 512 and 2048 in Table III.

For decreasing monomial codes at length $n = 2^m$, where m is an even number, we make the following conjecture about their largest mixing factors based on numerical observation. The conjectured upper bounds for decreasing monomial codes at length 16, 64, 256, 1024 are listed in Table IV.

TABLE III
MIXING FACTORS OF SELF-DUAL REED-MULLER CODES

code length n	8	32	128	512	2048
mixing factor	1	9	49	225	961

TABLE IV
CONJECTURED UPPER BOUNDS FOR DECREASING MONOMIAL CODES WITH RATES $\le 1/2$

code length n	16	64	256	1024
mixing factor $\le$	2	18	98	450

TABLE V
A TABLE ILLUSTRATING THE POSITIONS OF g AND g' IN THE PROOF OF THEOREM 4

$[\![f]\!]$	$[f]$	f
0	$(\underbrace{0,0,\cdots,0,0}_{2t+1})$	$x_0x_1x_2\cdots x_{2t}$
1	$(0,0,\cdots,0,1)$	$x_1x_2\cdots x_{2t}$
$\vdots$	$\vdots$	$\vdots$
$2^{2t+1}-2^{t+1}$	$(\underbrace{1,\cdots,1}_{t},\underbrace{0,\cdots,0,0}_{t+1})$	$g = x_0x_1x_2\cdots x_t$
$2^{2t+1}-2^{t+1}+1$	$(\underbrace{1,\cdots,1}_{t},0,\cdots,0,1)$	$g' = x_1x_2\cdots x_t$
$\vdots$	$\vdots$	$\vdots$
$2^{2t+1}-2$	$(1,1,\cdots,1,0)$	x_0
$2^{2t+1}-1$	$(1,1,\cdots,1,1)$	1

Conjecture 1: Let $\mathbb{C}$ be an (n, k) decreasing monomial code, with $n = 2^m$, $m = 2t$, and dimension $k \le n/2$, then

$$\mathrm{MF}(\mathbb{C}) \le 2^{2t-1} - 2^{t+1} + 2$$

where the equality is achievable.

The rest of this section is devoted to the proof of Theorem 4.

Proof of Theorem 4: It can be verified by exhaustive search that Theorem 4 holds when $t = 1$ and $t = 2$. So hereforth, we focus on proving the theorem when $t \ge 3$. In this proof we use Table V to help illustrate our arguments. First we show self-dual Reed-Muller codes achieve the equality in (13).

Claim 1. Let $\mathbb{C}$ be the self-dual Reed-Muller code of length 2^{2t+1}, then $\mathrm{MF}(\mathbb{C}) = 2^{2t} - 2^{t+1} + 1$.

Proof. Let $\mathcal{I}$ be set of monomials generating $\mathbb{C}$, Then $\mathcal{I}$ contains all monomials of degree less or equal to t. Referring to Table V, we have $\tau(\mathbb{C}) = 2^{2t+1} - 2^{t+1}$. Thus from equation (12), we have

$$\mathrm{MF}(\mathbb{C}) = 2^{2t} - 2^{t+1} + 1.$$

Then we focus on the following claim, which states that if the mixing factor of the code is at least $2^{2t} - 2^{t+1} + 1$, then the code has to be the self-dual Reed-Muller code.

Claim 2. Let $\mathbb{C}$ be a decreasing monomial code of length $n = 2^{2t+1}$ and dimension $k \le n/2$. If

$$\mathrm{MF}(\mathbb{C}) \ge 2^{2t} - 2^{t+1} + 1,$$

then $\mathbb{C}$ can only be the self-dual Reed-Muller code.

Now it suffices to prove Claim 2, since it is clear that Theorem 4 follows if we combine Claim 1 and Claim 2. Hereforth, we denote $g = x_0x_1x_2\cdots x_t$ as the monomial with $[\![g]\!] = 2^{2t+1} - 2^{t+1}$, and denote $g' = x_1x_2\cdots x_t$ as the monomial with $[\![g']\!] = [\![g]\!]+1$. If we list out all the monomials in $\mathcal{M}_{2t+1}$ following their row indices, the positions of g and g' in this list are shown in Table V.

Our proof for Claim 2 relies on the following three claims.

Claim 3. Let $\mathbb{C}$ be a decreasing monomial code of length $n = 2^{2t+1}$, and frozen index set $\mathcal{F}$. If $\tau(\mathbb{C}) \geq [\![g]\!]$, then $[\![g]\!] \in \mathcal{F}$.

Proof. Observe from Table V that for any monomial h with $[\![h]\!] \geq [\![g]\!]$, h is a divisor of g, which gives us $h \preccurlyeq g$. Therefore, if $\tau(\mathbb{C}) \geq [\![g]\!]$, it follows from Theorem 3 that $[\![g]\!] \in \mathcal{F}$.

Claim 4. Let $\mathbb{C}$ be a decreasing monomial code of length $n = 2^{2t+1}$, and frozen index set $\mathcal{F}$. If $\tau(\mathbb{C}) \geq [\![g']\!]$, then $[\![g']\!] \in \mathcal{F}$.

Proof. It can be observed from Table V that, for any monomial h with $[\![h]\!] \geq [\![g']\!]$, we have $h \preccurlyeq g'$. Therefore, similar to the proof for Claim 3, if $\tau(\mathbb{C}) \geq [\![g']\!]$, it follows from Theorem 3 that $[\![g']\!] \in \mathcal{F}$.

Claim 5. Let $\mathbb{C}$ be a decreasing monomial code of length $n = 2^{2t+1}$ and dimension $k \leq n/2$. If

$$\mathrm{MF}(\mathbb{C}) \geq 2^{2t} - 2^{t+1} + 1 \quad \text{and} \quad \tau(\mathbb{C}) = [\![g]\!],$$

then $\mathbb{C}$ is the self-dual Reed-Muller code.

Proof. Since for any monomial h with $\deg h \geq t+1$, we have $g \preccurlyeq h$, it follows from Theorem 3 that $[\![h]\!] \in \mathcal{F}$ for any monomial h with degree at least $t+1$. So $\mathbb{C}$ is a subcode of the self-dual Reed-Muller code. On the other hand, in view of equation (12), the dimension of the code is at least

$$k = \mathrm{MF}(\mathbb{C}) + (n-1) - \tau(\mathbb{C}) \geq n/2$$

Thus $\mathbb{C}$ can only be the self-dual Reed-Muller code itself.

At this point, we are ready to prove Claim 2. We will first show that given the conditions in Claim 2, g has to be frozen. Moreover, we will then show that the last frozen index of the code has to be exactly $[\![g]\!]$.

Proof of Claim 2. First from equation (12), we have

$$\tau(\mathbb{C}) = \mathrm{MF}(\mathbb{C}) + (n-k) - 1 \geq 2^{2t+1} - 2^{t+1}$$

So the last frozen index of $\mathbb{C}$ is at least $[\![g]\!]$. Then we show that $\tau(\mathbb{C}) > [\![g]\!]$ leads to a contradiction. Assuming $\tau(\mathbb{C}) > [\![g]\!]$, we have $[\![g]\!] \in \mathcal{F}$ and $[\![g']\!] \in \mathcal{F}$ following Claim 3 and Claim 4, respectively. Now we count the number of monomials having row indices in $\mathcal{F}$. First for any h with $\deg h \geq t+1$, we have $h \succcurlyeq g$. Thus it follows from Theorem 3 that $[\![h]\!] \in \mathcal{F}$ for all h with $\deg h \geq t+1$. The number of those monomials can be counted as

$$\sum_{i=t+1}^{2t+1} \binom{2t+1}{i} = 2^{2t}$$

Then for any degree-t monomial h without x_0, we have $h \succcurlyeq g'$, which gives us $[\![h]\!] \in \mathcal{F}$ following Theorem 3. The number of those monomials can be counted as $\binom{2t}{t}$. Therefore, the number of frozen indices of $\mathbb{C}$ is at least

$$|\mathcal{F}| \geq 2^{2t} + \binom{2t}{t}$$

This gives

$$|\mathcal{A}| = n - |\mathcal{F}| \leq 2^{2t} - \binom{2t}{t}$$

But that contradicts $\mathrm{MF}(\mathbb{C}) \geq 2^{2t} - 2^{t+1} + 1$, since

$$2^{2t} - \binom{2t}{t} < 2^{2t} - 2^{t+1} + 1$$

for all $t \geq 3$.

Since $\tau(\mathbb{C}) > [\![g]\!]$ leads to a contradiction, we can only have $\tau(\mathbb{C}) = [\![g]\!]$. Thus it follows from Claim 5 that $\mathbb{C}$ can only be the self-dual Reed-Muller code. ■

VI. Reducing Computation Complexity using a Subgroup of LTA

As a family of codes including polar codes, decreasing monomial codes have a large automorphism group. It was first shown that the automorphism group of decreasing monomial codes includes the lower triangular affine group (LTA) in [13]. Recently, this result has been extended to the block lower triangular affine group (BLTA) [20]. In this section, we look into the algebraic properties of decreasing monomial codes, and focus on a subgroup of LTA. We prove that this subgroup acts transitively on certain subsets of decreasing monomial codes. This result implies that those subsets share the same weight distribution, allowing us to drastically reduce the complexity of our approach.

A. Lower Triangular Affine Groups and Their Group Action

We start by reviewing the definition for the lower triangular affine group, and how it acts on polynomials. Henceforth, binary $m \times m$ matrices are denoted by $\mathbb{F}_2^{m\times m}$, and m-tuples in $\mathbb{F}_2^m$ are treated as column vectors. Following the notation in [13], we denote the affine transformation $\boldsymbol{x} \mapsto A\boldsymbol{x} + \boldsymbol{b}$ over $\mathbb{F}_2^m$ by a pair $(A, \boldsymbol{b})$, where $A \in \mathbb{F}_2^{m\times m}$ and $\boldsymbol{b} \in \mathbb{F}_2^m$.

Definition 7: The *lower triangular affine group*, denoted as $\mathrm{LTA}(m,2)$, consists of all affine transformations $(A, \boldsymbol{b})$, where $A \in \mathbb{F}_2^{m\times m}$ is a non-singular lower triangular square matrix, and $\boldsymbol{b} \in \mathbb{F}_2^m$.

The group action of $\mathrm{LTA}(m,2)$ on the polynomial ring $\mathcal{R}_m$ can be defined as follows. For an affine transformation $(A, \boldsymbol{b}) \in \mathrm{LTA}(m,2)$ with $A = (a_{ij})$, and a polynomial

$p \in \mathcal{R}_m$, we denote by $(A, \boldsymbol{b}) \cdot p$ the action of $(A, \boldsymbol{b})$ on p, where each monomial x_i in p is replaced by another monomial y_i defined as

$$y_i = \sum_{j=0}^{m-1} a_{ij} x_j + b_i.$$

For monomials, the following expansion after the group action of $\mathrm{LTA}(m, 2)$ can be observed, which follows directly from Definition 7.

Proposition 3: Let $(A, \boldsymbol{b}) \in \mathrm{LTA}(m, 2)$ and $f \in \mathcal{M}_m$, then $(A, \boldsymbol{b}) \cdot f$ can be expanded as

$$(A, \boldsymbol{b}) \cdot f = f + \sum_{g \in \mathcal{M}_m :\, g \prec f} u_g \cdot g, \tag{14}$$

where $u_g \in \{0, 1\}$ for all g.

Here is another way to view the action by the affine transformations. Recall that the evaluation $\mathrm{ev}(p)$ of a polynomial $p \in \mathcal{R}_m$ is a vector that consists of $p(\boldsymbol{x})$ over all $\boldsymbol{x} \in \mathbb{F}_2^m$. Since every affine transformation $(A, \boldsymbol{b})$ is a bijection on $\mathbb{F}_2^m$, the evaluation $\mathrm{ev}((A, \boldsymbol{b}) \cdot p)$ can be obtained from $\mathrm{ev}(p)$ by permuting its coordinates. Denote the action of $(A, \boldsymbol{b})$ on a polynomial evaluation as

$$(A, \boldsymbol{b}) \cdot \mathrm{ev}(p) = \mathrm{ev}((A, \boldsymbol{b}) \cdot p),$$

we can then view this action as a permutation on the coordinates of $\mathrm{ev}(p)$. In particular, vector $(A, \boldsymbol{b}) \cdot \mathrm{ev}(p)$ and vector $\mathrm{ev}(p)$ have the same Hamming weight.

In the work by Bardet, Dragoi, Otmani, and Tillich, they show that the automorphism group of decreasing monomial codes over m variables includes the lower triangular affine group $\mathrm{LTA}(m, 2)$ [13, Theorem 2].

B. A Subgroup of $\mathrm{LTA}(m, 2)$

In the main theorem of this section, we consider a subgroup of $\mathrm{LTA}(m, 2)$, denoted $\mathrm{LTA}(m, 2)_f$, that we associate with a given monomial f. This subgroup was introduced in [13], where it was used to analyze and count the minimum weight codewords of decreasing monomial codes.

Definition 8: Let $f \in \mathcal{M}_m$. The subgroup $\mathrm{LTA}(m, 2)_f$ of $\mathrm{LTA}(m, 2)$ associated with the monomial f is defined as

$$\mathrm{LTA}(m, 2)_f \triangleq \{(A, \boldsymbol{b}) \in \mathrm{LTA}(m, 2) \mid A \in M_f, \boldsymbol{b} \in B_f\},$$

where

$$M_f = \{(a_{ij}) \in \mathbb{F}_2^{m \times m} \mid \forall\, i > j, \quad a_{ij} = 0 \text{ if } i \notin \mathrm{ind}(f) \text{ or } j \in \mathrm{ind}(f)\}$$

and

$$B_f = \{\boldsymbol{b} \in \mathbb{F}_2^m \mid b_i = 0 \text{ if } i \notin \mathrm{ind}(f)\}$$

Example 5: Consider $\mathrm{LTA}(5, 2)$. For $f = x_0 x_3 x_4 \in \mathcal{M}_5$, we have

$$M_f = \{(a_{ij}) \in \mathbb{F}_2^{5 \times 5} \mid \forall\, i > j, \quad a_{ij} = 0 \text{ if } i \notin \{0, 3, 4\} \text{ or } j \in \{0, 3, 4\}\}$$

and

$$B_f = \{\boldsymbol{b} \in \mathbb{F}_2^m \mid b_i = 0 \text{ if } i \notin \{0, 3, 4\}\}$$

Therefore, the affine transformations $(A, \boldsymbol{b})$ in the subgroup $\mathrm{LTA}(5, 2)_f$ have the form:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & a_{3,1} & a_{3,2} & 1 & 0 \\ 0 & a_{4,1} & a_{4,2} & 0 & 1 \end{pmatrix}, \quad \text{and} \quad b = \begin{pmatrix} b_0 \\ 0 \\ 0 \\ b_3 \\ b_4 \end{pmatrix},$$

where $a_{3,1}$, $a_{3,2}$, $a_{4,1}$, $a_{4,2}$, b_0, b_3, b_4 can take any value in $\{0, 1\}$. There are 2^7 such affine transformations, so the order of $\mathrm{LTA}(2, 4)_f$ is $2^7 = 128$.

C. One-Variable Descendance Relation

We also introduce a new relation on the monomials for our main theorem of this section. Henceforth, whenever we write $f = g x_i$ for two monomials f and g, we assume $i \notin \mathrm{ind}(g)$.

Definition 9: For $f, g \in \mathcal{M}_m$, we say g is a *one-variable descendant* of f, and write $g \prec_1 f$ if either one of the following holds:

1) $f = h x_i$ and $g = h x_j$ for some monomial h with $j < i$.
2) $f = g x_i$

Compared with the partial order in Definition 5, this one-variable descendance relation is a more restricted relation in the sense that, the two involved monomials can only differ by *one* variable. We remark that this one-variable descendance relation is only a relation, but not a partial order on the monomials. The following example shows that this new relation is not transitive.

Example 6: For monomials in $\mathcal{M}_4$, we have

$$x_0 x_2 \prec_1 x_0 x_1 x_2, \quad \text{and} \quad x_0 x_1 x_2 \prec_1 x_0 x_1 x_3,$$

but $x_0 x_2$ is not a one-variable descendant of $x_0 x_1 x_3$.

D. The Main Theorem: A Transitive Group Action

Now we are ready to present the main theorem of this section.

Theorem 5: Let $\mathbb{C}(\mathcal{I})$ be a decreasing monomial code generated by $\mathcal{I} \in \mathcal{M}_m$, and let f be the monomial in $\mathcal{I}$ with the smallest row index:

$$f = \underset{g \in \mathcal{I}}{\operatorname{argmin}}\, [\![g]\!]$$

We partition $\mathcal{I}$ into the following disjoint union

$$\mathcal{I} = \{f\} \cup \mathcal{S} \cup \mathcal{T},$$

where $\mathcal{S}$ consists of all one-variable descendant of f with row indices smaller than $\tau(\mathbb{C})$, and $\mathcal{T}$ contains the rest of the monomials in $\mathcal{I}$:

$$\mathcal{S} = \{h \in \mathcal{I} \mid h \prec_1 f \text{ and } [\![h]\!] < \tau(\mathbb{C})\},$$

and

$$\mathcal{T} = \mathcal{I} \setminus (\{f\} \cup \mathcal{S}).$$

	$x_0x_1x_2x_3x_4$	u_0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_1x_2x_3x_4$	u_1	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_0x_2x_3x_4$	u_2	1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_2x_3x_4$	u_3	1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	
	$x_0x_1x_3x_4$	u_4	1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_1x_3x_4$	u_5	1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	
$f \rightarrow$	$x_0x_3x_4$	u_6	1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
$\mathcal{S}$ $\rightarrow$	x_3x_4	u_7	1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0	
	$x_0x_1x_2x_4$	u_8	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_1x_2x_4$	u_9	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	
$\rightarrow$	$x_0x_2x_4$	u_{10}	1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_2x_4	u_{11}	1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0	
$\rightarrow$	$x_0x_1x_4$	u_{12}	1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_1x_4	u_{13}	1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0	
$\rightarrow$	x_0x_4	u_{14}	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_4	u_{15}	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	
	$x_0x_1x_2x_3$	u_{16}	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_1x_2x_3$	u_{17}	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	$x_0x_2x_3$	u_{18}	1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_2x_3	u_{19}	1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0	
	$x_0x_1x_3$	u_{20}	1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	generate $\mathbb{C}(\mathcal{T})$
	x_1x_3	u_{21}	1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0	
	x_0x_3	u_{22}	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_3	u_{23}	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0	
	$x_0x_1x_2$	u_{24}	1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_1x_2	u_{25}	1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	
	x_0x_2	u_{26}	1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_2	u_{27}	1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0	
	x_0x_1	u_{28}	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	x_1	u_{29}	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0	
	x_0	u_{30}	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
	1	u_{31}	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	

Fig. 6. Polar Transformation matrix G_{32} in Example 7

Then the group action of subgroup $\text{LTA}(m,2)_f$ on the set $\mathcal{X}$ is transitive, where $\mathcal{X}$ is the set consisting of cosets of $\mathbb{C}(\mathcal{T})$ defined as follows

$$\mathcal{X} = \left\{ \text{ev}(f) + \sum_{h\in\mathcal{S}} u_h \cdot \text{ev}(h) + \mathbb{C}(\mathcal{T}) \,\middle|\, \forall\, h \in \mathcal{S},\ u_h \in \{0,1\} \right\} \tag{15}$$

Therefore, all the cosets of $\mathbb{C}(\mathcal{T})$ in $\mathcal{X}$ have the same weight distribution.

Before proving this theorem, we illustrate Theorem 5 with an example, and show how we can use this theorem to reduce the complexity when computing the weight distribution of decreasing monomial codes.

Example 7: Consider a (32,24) decreasing monomial code $\mathbb{C}$ specified by the frozen index set $\mathcal{F} = \{0,1,2,3,4,5,8,16\}$. The monomials corresponding to the rows in G_{32} are shown in Figure 6, where the information bits are highlighted in red, orange and blue, and the frozen bits are black. Code $\mathbb{C}$ has last frozen index $\tau(\mathbb{C}) = 16$, and mixing factor $\text{MF}(\mathbb{C}) = 9$.

Illustrating Theorem 5

Let $f = x_0x_3x_4$ be the monomial with the smallest row index in $\mathcal{I}$. Then $\mathcal{I}$ can be partitioned as

$$\mathcal{I} = \{f\} \cup \mathcal{S} \cup \mathcal{T},$$

where $\mathcal{S}$ consists of four of the one-variable descendants of f with row indices smaller than $\tau(\mathbb{C}) = 16$, and $\mathcal{T}$ consists of the rest of the monomials in $\mathcal{I}$:

$$\mathcal{S} = \{x_3x_4, x_0x_2x_4, x_0x_1x_4, x_0x_4\},$$

$$\mathcal{T} = \{x_1x_2x_4, x_2x_4, x_1x_4, x_4, x_1x_2x_3, x_0x_2x_3, \cdots, x_0, 1\}$$

As shown in Figure 6, the monomials in $\mathcal{S}$ are colored in red, the monomials in $\mathcal{T}$ are colored in orange and in blue, and the subcode $\mathbb{C}(\mathcal{T})$ is generated by the gray rows in G_{32}.

Then, set $\mathcal{X}$ is defined to consist of 16 cosets of $\mathbb{C}(\mathcal{T})$ in the form

$$\text{ev}(f) + u_1 \cdot \text{ev}(x_3x_4) + u_2 \cdot \text{ev}(x_0x_2x_4) + u_3 \cdot \text{ev}(x_0x_1x_4) + u_4 \cdot \text{ev}(x_0x_4) + \mathbb{C}(\mathcal{T}),$$

where u_1, u_2, u_3, u_4 are four coefficients that can take any value in $\{0,1\}$.

According to Theorem 5, the subgroup $\text{LTA}(5,2)_f$ acts transitively on $\mathcal{X}$. Since the group action of the affine transformations in $\text{LTA}(5,2)_f$ can be viewed as permutations on the codeword coordinates, we can conclude that all 16 cosets in $\mathcal{X}$ have the same weight distribution.

Computing the Weight Distribution

If we directly apply Algorithm 2 to compute the weight distribution of $\mathbb{C}$, we need to compute the weight enumerating function of 2^9 polar cosets. Now we show how we can reduce this number using Theorem 5.

We start by partitioning code $\mathbb{C}$ into two parts according to u_6. Let $\mathbb{C}\{u_6 = 1\}$ denote the subset of $\mathbb{C}$ where u_6 is fixed to be 1, and let $\mathbb{C}\{u_6 = 0\}$ denote the subcode of $\mathbb{C}$ where u_6 is fixed to be 0. Then

$$\mathbb{C} = \mathbb{C}\{u_6 = 1\} \cup \mathbb{C}\{u_6 = 0\},$$

and both $\mathbb{C}\{u_6 = 1\}$ and $\mathbb{C}\{u_6 = 0\}$ can be represented as disjoint unions of 2^8 polar cosets. Next, we compute the weight distribution of $\mathbb{C}\{u_6 = 1\}$ and $\mathbb{C}\{u_6 = 0\}$ separately.

For $\mathbb{C}\{u_6 = 1\}$, observe that

$$\mathbb{C}\{u_6 = 1\} = \bigcup_{X\in\mathcal{X}} X$$

This article has been accepted for publication in IEEE Transactions on Information Theory. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIT.2023.3243191

TABLE VI
THE COMPLEXITY REDUCTION AMOUNTS IN EXAMPLE 7

components	complexity	reduced complexity
$\mathbb{C}\{u_6 = 1\}$	$2^8 = 256$	$2^4 = 16$
$\mathbb{C}\{\cdots, u_7 = 1\}$	$2^7 = 128$	$2^3 = 8$
$\mathbb{C}\{\cdots, u_9 = 1\}$	$2^6 = 64$	$2^2 = 4$
$\mathbb{C}\{\cdots, u_{10} = 1\}$	$2^5 = 32$	$2^2 = 4$
$\mathbb{C}\{\cdots, u_{11} = 1\}$	$2^4 = 16$	$2^1 = 2$
$\mathbb{C}\{\cdots, u_{12} = 1\}$	$2^3 = 8$	$2^1 = 2$
$\mathbb{C}\{\cdots, u_{13} = 1\}$	$2^2 = 4$	$2^0 = 1$
$\mathbb{C}\{\cdots, u_{14} = 1\}$	$2^1 = 2$	$2^0 = 1$
$\mathbb{C}\{\cdots, u_{15} = 1\}$	$2^0 = 1$	$2^0 = 1$
$\mathbb{C}\{\cdots, u_{15} = 0\}$	$2^0 = 1$	$2^0 = 1$
$\mathbb{C}$	$2^9 = 512$	40

By Theorem 5, all cosets in $\mathcal{X}$ have the same weight distribution. Thus to get the weight distribution for $\mathbb{C}\{u_6 = 1\}$, it suffices to first compute the weight distribution of a single coset in $\mathcal{X}$ using Algorithm 2, and then multiply it by $|\mathcal{X}| = 2^4$. This reduces the number of polar cosets that we need to evaluate for $\mathbb{C}\{u_6 = 1\}$ from 2^8 down to 2^4.

After that we consider $\mathbb{C}\{u_6 = 0\}$. Since the subcode $\mathbb{C}\{u_6 = 0\}$ is also a decreasing monomial code itself, we can again partition $\mathbb{C}\{u_6 = 0\}$ into two parts according to u_7:

$$\mathbb{C}\{u_6 = 0\} = \mathbb{C}\{u_6 = 0, u_7 = 1\} \cup \mathbb{C}\{u_6 = 0, u_7 = 0\},$$

and apply Theorem 5 to reduce the complexity for the second term $\mathbb{C}\{u_6 = 0, u_7 = 1\}$.

By repeating this procedure, code $\mathbb{C}$ can be unfolded as follows

$$\begin{aligned}\mathbb{C} = &\ \mathbb{C}\{u_6 = 1\} \\ &\cup \mathbb{C}\{u_6 = 0, u_7 = 1\} \\ &\cup \mathbb{C}\{u_6 = 0, u_7 = 0, u_9 = 1\} \\ &\cup \cdots \\ &\cup \mathbb{C}\{u_6 = 0, u_7 = 0, \cdots, u_{14} = 0, u_{15} = 1\} \\ &\cup \mathbb{C}\{u_6 = 0, u_7 = 0, \cdots, u_{14} = 0, u_{15} = 0\},\end{aligned}$$

and Theorem 5 allows us to reduce the number of polar cosets that we need to evaluate for each of those components. The amount of complexity reduction for the components, and the total amount of complexity reduction for $\mathbb{C}$ are shown in Table VI. In Table VI, the second column shows the number of polar cosets in each component, and the third column shows the number of polar cosets that we need to evaluate after applying Theorem 5. We also show the computed weight distribution for $\mathbb{C}$ in this example in Table VII, where the unlisted A_d equals to zero.

E. Proof of Theorem 5

The rest of this section is devoted to the proof of Theorem 5. First let us introduce more notations. For any polynomial $p \in \mathcal{R}_m$, we can expand it and express p as a sum of monomials in $\mathcal{M}_m$ as

$$p = \sum_{q \in \mathcal{M}_m} u_q \cdot q, \tag{16}$$

TABLE VII
WEIGHT DISTRIBUTION OF $\mathbb{C}$ IN EXAMPLE 7

d	A_d	d	A_d	d	A_d
0	1	12	1768424	22	503424
4	472	14	3668224	24	83164
6	6272	16	4717254	26	6272
8	83164	18	3668224	28	472
10	503424	20	1768424	32	1

where $u_q \in \{0, 1\}$ are the coefficients. For each monomial q, we denote the coefficient u_q in this expansion of p by $\langle p \rangle_q$. Using this notation, equation (16) can be written as

$$p = \sum_{q \in \mathcal{M}_m} \langle p \rangle_q \cdot q,$$

We start our proof by establishing a few lemmas. First, we consider the group action of an affine transformation $(A, \boldsymbol{b})$ in the subgroup $\mathrm{LTA}(m, 2)_f$ on f itself. The following lemma states that the coefficient of a monomial $h \in \mathcal{S}$ in the expansion of $(A, \boldsymbol{b}) \cdot f$ can actually be determined by an entry in $(A, \boldsymbol{b})$.

Lemma 2: In Theorem 5, let $(A, \boldsymbol{b}) \in \mathrm{LTA}(m, 2)_f$ with $A = (a_{ij})$, and $h \in \mathcal{S}$, then the coefficient of h in the expansion of $(A, \boldsymbol{b}) \cdot f$ equals to an entry in $(A, \boldsymbol{b})$. More precisely,

- if $f = qx_s$ and $h = qx_t$ for some monomial q with $t < s$, then $\langle (A, \boldsymbol{b}) \cdot f \rangle_h = a_{st}$;
- if $f = hx_s$, then $\langle (A, \boldsymbol{b}) \cdot f \rangle_h = b_s$.

Proof: First, f can be written as

$$f = \prod_{i \in \mathrm{ind}(f)} x_i$$

Consider the action of $(A, \boldsymbol{b}) \in \mathrm{LTA}(m, 2)$ on f. According to Definition 8, each monomial x_i in f will be replaced by

$$y_i = x_i + \sum_{j<i:\ j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i$$

Therefore, $(A, \boldsymbol{b}) \cdot f$ can be written as a product of ℓ linear terms, where $\ell = \deg f$:

$$(A, \boldsymbol{b}) \cdot f = \prod_{i \in \mathrm{ind}(f)} \left(x_i + \sum_{j<i:\ j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i \right) \tag{17}$$

Given that $h \in \mathcal{S}$ is a one-variable descendant of f, we now verify this lemma by discussing the following two cases for h:

- Case 1: $f = qx_s$ and $h = qx_t$ for some monomial q with $t < s$.
  It can be observed that when we expand the right hand side of equation (17), there is only one way to generate the term h, corresponding to choosing $a_{st}x_t$ from the linear term led by x_s, and choosing the leading x_i for the rest of the linear terms. Thus $\langle (A, \boldsymbol{b}) \cdot f \rangle_h = a_{st}$.
- Case 2: $f = hx_s$.
  Similarly, it can be observed that when we expand the right hand side of equation (17), there is only one way to generate the term h, corresponding to choosing b_s from

the linear term led by x_s, and choosing the leading x_i for the rest of the linear terms. Thus $\langle (A,\boldsymbol{b}) \cdot f \rangle_h = b_s$. ■

Next, we consider the group action of an $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$ on a monomial $g \in \mathcal{T}$. The following lemma states that, the coefficient of a monomial $h \in \mathcal{S}$ in the expansion of $(A,\boldsymbol{b}) \cdot g$ always equals to zero.

Lemma 3: In Theorem 5, let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$, $h \in \mathcal{S}$, and $g \in \mathcal{T}$, then the coefficient of h in the expansion of $(A,\boldsymbol{b}) \cdot g$ is zero. In other words, $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 0$.

Proof: Consider the action of $(A,\boldsymbol{b})$ on g. According to Definition 8, the monomials in g will change as follows:

- Every x_i with $i \in \mathrm{ind}(g) \cap \mathrm{ind}(f)$ will be replaced by
$$y_i = x_i + \sum_{j<i,\ j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i$$
- Every x_i with $i \in \mathrm{ind}(g) \setminus \mathrm{ind}(f)$ will be replaced by $y_i = x_i$, and thus remain unchanged.

So after the action by $(A,\boldsymbol{b})$ on
$$g = \prod_{i \in \mathrm{ind}(g)} x_i,$$
we have
$$(A,\boldsymbol{b}) \cdot g = \underbrace{\left(\prod_{i \in \mathrm{ind}(g) \setminus \mathrm{ind}(f)} x_i \right)}_{(a)} \cdot \underbrace{\left(\prod_{i \in \mathrm{ind}(g) \cap \mathrm{ind}(f)} \left(x_i + \sum_{j<i,\ j \notin \mathrm{ind}(f)} a_{ij} x_j + b_i \right) \right)}_{(b)} \quad (18)$$

If $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 1$, then h should appear if we expand the right hand side of (18). Since $h \in \mathcal{S}$ is a one-variable descendant of f, according to Definition 9, we have the following two possible cases for the relation between h and f. Next, we show that if $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 1$, a contradiction can be drawn in both cases.

- Case 1: $f = qx_s$ and $h = qx_t$ for some monomial q with $t < s$.
  If h appears in the expansion of the right hand side of (18), then we break it into two cases depending on where the x_t in h comes from.
  - If the x_t in h comes from parenthesis (a) in (18), then we must have $\mathrm{ind}(g) \setminus \mathrm{ind}(f) = t$, and $\mathrm{ind}(q) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$. Since q is a divisor of f, for $\mathrm{ind}(q) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$ to be true, we can either have
  $$\mathrm{ind}(q) = \mathrm{ind}(g) \cap \mathrm{ind}(f) \quad \Rightarrow \quad g = h$$
  which is a contradiction since g and h are distinct, or
  $$\mathrm{ind}(q) \cup \{s\} = \mathrm{ind}(g) \cap \mathrm{ind}(f) \quad \Rightarrow \quad g = qx_t x_s,$$
  which is also a contradiction since $[\![g]\!] > [\![f]\!]$.
  - If the x_t in h comes from parenthesis (b) in (18), then we must have $\mathrm{ind}(g) \setminus \mathrm{ind}(f) = \emptyset$, and $\mathrm{ind}(q) \cup \{s\} \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$, giving us
  $$\mathrm{ind}(g) = \mathrm{ind}(f) \quad \Rightarrow \quad g = f,$$
  which is a contradiction since g and f are distinct.
- Case 2: $f = hx_s$.
  If h appears in the expansion of the right hand side of (18), then we must have $\mathrm{ind}(g) \setminus \mathrm{ind}(f) = \emptyset$, and $\mathrm{ind}(h) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$. Since h is a divisor of f, for $\mathrm{ind}(h) \subseteq \mathrm{ind}(g) \cap \mathrm{ind}(f)$ to be true, we can either have
  $$\mathrm{ind}(h) = \mathrm{ind}(g) \cap \mathrm{ind}(f) \quad \Rightarrow \quad g = h,$$
  which is a contradiction since g and h are distinct, or
  $$\mathrm{ind}(h) \cup \{s\} = \mathrm{ind}(g) \cap \mathrm{ind}(f) \quad \Rightarrow \quad g = f$$
  which is also a contradiction since g and f are distinct.

Therefore, in both Case 1 and Case 2, $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 1$ leads to a contradiction. Thus we can only have $\langle (A,\boldsymbol{b}) \cdot g \rangle_h = 0$. ■

Using Lemma 3, we can prove that subcode $\mathbb{C}(\mathcal{T})$ is invariant under $\mathrm{LTA}(m,2)_f$, as stated in the following lemma.

Lemma 4: In Theorem 5, subcode $\mathbb{C}(\mathcal{T})$ is invariant under $\mathrm{LTA}(m,2)_f$.

Proof: Let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$. The group action by $(A,\boldsymbol{b})$ can be viewed as a permutation on the codeword coordinates, so $(A,\boldsymbol{b})$ acting on $\mathbb{C}(\mathcal{T})$ will generate another subspace with the same dimension as $\mathbb{C}(\mathcal{T})$. Since $\mathbb{C}(\mathcal{T})$ is generated by the monomials in $\mathcal{T}$, to prove this claim, it suffices to prove that for any $g \in \mathcal{T}$, we have $(A,\boldsymbol{b}) \cdot \mathrm{ev}(g) \in \mathbb{C}(\mathcal{T})$.

Let $(A,\boldsymbol{b}) \in \mathrm{LTA}(m,2)_f$ and $g \in \mathcal{T}$. First, it follows from Proposition 3 that
$$(A,\boldsymbol{b}) \cdot g = g + \sum_{g' \in \mathcal{M}_m:\ g' \prec g} u'_g \cdot g', \quad (19)$$
where $u'_g \in \{0,1\}$ are coefficients for all g'. Then, since $\mathcal{I}$ is the generating set of a decreasing monomial code, from Theorem 3 we know all g' with $g' \prec g$ are in $\mathcal{I}$. Hence (19) can be written as
$$(A,\boldsymbol{b}) \cdot g = g + \sum_{g' \in \mathcal{I}:\ g' \prec g} u'_g \cdot g'. \quad (20)$$

Recall f is the monomial with the smallest row index in $\mathcal{I}$, so it follows from Lemma 1 that $f \not\preccurlyeq g$. Also, Lemma 3 tells us that in (20), $u_h = 0$ for all $h \in \mathcal{S}$. Since $\mathcal{I} = \{f\} \cup \mathcal{S} \cup \mathcal{T}$, (20) becomes
$$(A,\boldsymbol{b}) \cdot g = g + \sum_{g' \in \mathcal{T}:\ g' \prec g} u'_g \cdot g'$$
Therefore, any $(A,\boldsymbol{b}) \cdot g$ with $g \in \mathcal{T}$ can be generated by the monomials in $\mathcal{T}$. This finishes the proof of this lemma. ■

At this point, we are ready to put everything together and prove Theorem 5. Take $X_0 = \mathrm{ev}(f) + \mathbb{C}(\mathcal{T})$ to be a coset in $\mathcal{X}$. To prove that the group action of $\mathrm{LTA}(m,2)_f$ on $\mathcal{X}$ is transitive, it suffices to prove that the orbit of X_0 is the entire $\mathcal{X}$.

Let $(A, \boldsymbol{b})$ be an affine transformation in $\mathrm{LTA}(m, 2)_f$. If we consider the action of $(A, \boldsymbol{b})$ on f, It follows from Proposition 3 that

$$(A, \boldsymbol{b}) \cdot f = f + \sum_{h \in \mathcal{S}} u_h \cdot h + \sum_{g \in \mathcal{T}} u_g \cdot g \tag{21}$$

where $u_h = \langle (A, \boldsymbol{b}) \cdot f \rangle_h$ for each $h \in \mathcal{S}$, and $u_g = \langle (A, \boldsymbol{b}) \cdot f \rangle_g$ for each $g \in \mathcal{T}$. Therefore, if we look at the action of $(A, \boldsymbol{b})$ on X_0, we have

$$\begin{aligned}
&(A, \boldsymbol{b}) \cdot X_0 \\
&= \mathrm{ev}(f) + \sum_{h \in \mathcal{S}} u_h \cdot \mathrm{ev}(h) + \sum_{g \in \mathcal{T}} u_g \cdot \mathrm{ev}(g) + (A, \boldsymbol{b}) \cdot \mathbb{C}(\mathcal{T}) && (22) \\
&= \mathrm{ev}(f) + \sum_{h \in \mathcal{S}} u_h \cdot \mathrm{ev}(h) + \sum_{g \in \mathcal{T}} u_g \cdot \mathrm{ev}(g) + \mathbb{C}(\mathcal{T}) && (23) \\
&= \mathrm{ev}(f) + \sum_{h \in \mathcal{S}} u_h \cdot \mathrm{ev}(h) + \mathbb{C}(\mathcal{T}) && (24) \\
&= \mathrm{ev}(f) + \sum_{h \in \mathcal{S}} \langle (A, \boldsymbol{b}) \cdot f \rangle_h \cdot \mathrm{ev}(h) + \mathbb{C}(\mathcal{T}) && (25)
\end{aligned}$$

where

- in (22), we have $(A, \boldsymbol{b}) \cdot \mathbb{C}(\mathcal{T}) = \mathbb{C}(\mathcal{T})$, since $(A, \boldsymbol{b}) \in \mathrm{LTA}(m, 2)_f$, and $\mathbb{C}(\mathcal{T})$ is invariant under $\mathrm{LTA}(m, 2)_f$ from Lemma 4.
- in (23), we have

  $$\sum_{g \in \mathcal{T}} u_g \cdot \mathrm{ev}(g) + \mathbb{C}(\mathcal{T}) = \mathbb{C}(\mathcal{T}),$$

  since $\mathrm{ev}(g) \in \mathbb{C}(\mathcal{T})$ for all $g \in \mathcal{T}$.

In (25), according to Lemma 2, every $\langle (A, \boldsymbol{b}) \cdot f \rangle_h$ equals to an entry in $(A, \boldsymbol{b})$. Therefore, given any $X \in \mathcal{X}$, we can pick an $(A, \boldsymbol{b}) \in \mathrm{LTA}(m, 2)_f$ whose entries are chosen such that X can be generated by $(A, \boldsymbol{b}) \cdot X_0$. This proves that the orbit of X_0 is the entire set $\mathcal{X}$, which means the group action of $\mathrm{LTA}(m, 2)_f$ on the $\mathcal{X}$ is transitive. Since the action by affine transformations in $\mathrm{LTA}(m, 2)_f$ can be viewed as permutations on the codeword coordinates, all the cosets in $\mathcal{X}$ thus have the same weight distribution. This completes the proof.

VII. Applications of Our Algorithm

In this section, we present some application examples of our algorithm on decreasing monomial codes. For those codes, we also show how much the complexity of our approach can be reduced using the method from Section VI.

First, we present the weight distribution of a $(128, 64)$ polar code constructed based on the reliability sequence in the 5G technical specification [26] without CRC. This polar code has mixing factor 34, so if we directly apply Algorithm 2, the number of polar cosets that we need to evaluate equals 2^{34}. The code can be verified to be a decreasing monomial code, so this complexity can be reduced using the method from Section VI. After the complexity reduction, the number of polar cosets that we need to evaluate can be reduced to $39257360 \approx 2^{25.23}$. The computed weight distribution of this code is shown in Table VIII. For reference, computing this weight distribution takes less than two hours on a laptop computer.

TABLE VIII
Weight distribution of the (128,64) polar code constructed following the reliability sequence in 5G [26]

d	A_d
0	1
8	304
12	768
16	161528
20	4452096
24	166137744
28	8299319808
32	474588991516
36	19910428320256
40	555627871531568
44	9459383897458944
48	94101946507153608
52	550051775557674240
56	1920378732932218128
60	4051638142931561472
64	5194332067339587654
68	4051638142931561472
72	1920378732932218128
76	550051775557674240
80	94101946507153608
84	9459383897458944
88	555627871531568
92	19910428320256
96	474588991516
100	8299319808
104	166137744
108	4452096
112	161528
116	768
120	304
128	1

Then, we look at the (128,64) Reed-Muller code. Note that the weight distribution of this Reed-Muller code has already been computed by Sugino, Ienaga, Tokura and Kasami in [6]. This Reed-Muller code has mixing factor 49, so in our approach, the number of polar cosets that we need to evaluate in Algorithm 2 equals 2^{49}. If we apply the complexity reduction from Section VI, this number can be reduced to $49761365064 \approx 2^{35.53}$, which is a viable computation complexity that can be achieved. Since this self-dual Reed-Muller code has the largest mixing factor among all decreasing monomial codes with rate at most 1/2 at length 128. It is reasonable to expect that after we apply the complexity reduction from Section VI, the number of polar cosets that we need to evaluate for other decreasing monomial codes at length 128 will not be much larger than 2^{35}. Therefore, we believe that our approach allows us to compute the weight distribution of any decreasing monomial codes at length 128.

VIII. Conclusion

In this paper, we present a deterministic algorithm for computing the exact weight distribution of polar codes at length 128. First, we propose a recursive procedure for computing the weight distribution of polar cosets along arbitrary decoding path. Then, we show that any polar code can be represented as a disjoint union of polar cosets. Therefore, the entire weight distribution of the code can be obtained by first computing the weight distribution of all the polar cosets in this representation, and then taking the sum. However, the number of polar cosets in this representation grows exponentially with a parameter called mixing factor. To bound the complexity of our approach, we provide a bound on the mixing factor of polar codes being decreasing monomial codes. To further reduce this complexity, we study the algebraic structure of decreasing monomial codes, and prove that a subgroup of the lower triangular affine group acts transitively on certain subsets of decreasing monomial codes. This allows us to reduce the number of polar cosets that we need to evaluate in our approach. After the complexity reduction, our algorithm still has exponential complexity, but it is efficient enough to compute the weight distribution of any decreasing monomial codes at length 128.

References

[1] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[2] Y. Desaki, T. Fujiwara, and T. Kasami, "The weight distributions of extended binary primitive BCH codes of length 128," *IEEE Transactions on Information Theory*, vol. 43, no. 4, pp. 1364–1371, 1997.

[3] T. Fujiwara, Y. Desaki, T. Sugita, and T. Kasami, "The weight distributions of several extended binary primitive BCH codes of length 256," in *Proceedings of IEEE International Symposium on Information Theory*. June, 1997, p. 363.

[4] T. Fujiwara and T. Kasami, "The weight distribution of (256, k) extended binary primitive BCH code with $k \leq 63$, $k \geq 207$," Technical Report of IEICE, IT97, Tech. Rep., 1993.

[5] T. Fujiwara and T. Kusaka, "The weight distributions of the (256, k) extended binary primitive BCH codes with $k \leq 71$ and $k \geq 187$," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, pp. 1321–1328, 2021.

[6] M. Sugino, Y. Ienaga, N. Tokura, and T. Kasami, "Weight distribution of (128, 64) Reed-Muller code (corresp.)," *IEEE Transactions on Information Theory*, vol. 17, no. 5, pp. 627–628, 1971.

[7] T. Sugita, T. Kasami, and T. Fujiwara, "The weight distribution of the third-order Reed-Muller code of length 512," *IEEE Transactions on Information Theory*, vol. 42, no. 5, pp. 1622–1625, 1996.

[8] 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.

[9] J. A. MacWilliams, "A theorem on the distribution of weights in a systematic code," *Bell System Technical Journal*, vol. 42, no. 1, pp. 79–94, 1963.

[10] M. Xu, P. Chen, B. Bai, and S. Tong, "Distance spectrum and optimized design of concatenated polar codes," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*. October, 2017, pp. 1–6.

[11] M. Valipour and S. Yousefi, "On probabilistic weight distribution of polar codes," *IEEE communications letters*, vol. 17, no. 11, pp. 2120–2123, 2013.

[12] Q. Zhang, A. Liu, and X. Pan, "An enhanced probabilistic computation method for the weight distribution of polar codes," *IEEE Communications Letters*, vol. 21, no. 12, pp. 2562–2565, 2017.

[13] M. Bardet, V. Dragoi, A. Otmani, and J.-P. Tillich, "Algebraic properties of polar codes from a new polynomial formalism," in *Proceedings of IEEE International Symposium on Information Theory*. July, 2016, pp. 230–234.

[14] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Communications Letters*, vol. 16, no. 12, pp. 2044–2047, 2012.

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

[16] Z. Liu, K. Chen, K. Niu, and Z. He, "Distance spectrum analysis of polar codes," in *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. April, 2014, pp. 490–495.

[17] K. Niu, Y. Li, and W. Wu, "Polar codes: Analysis and construction based on polar spectrum," *arXiv preprint arXiv:1908.05889*, 2019.

[18] R. Polyanskaya, M. Davletshin, and N. Polyanskii, "Weight distributions for successive cancellation decoding of polar codes," *IEEE Transactions on Communications*, vol. 68, no. 12, pp. 7328–7336, 2020.

[19] P. Trifonov and V. Miloslavskaya, "Polar codes with dynamic frozen symbols and their decoding by directed search," in *IEEE Information Theory Workshop (ITW)*. September, 2013, pp. 1–5.

[20] M. Geiselhart, A. Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "On the automorphism group of polar codes," in *Proceedings of IEEE International Symposium on Information Theory*. July, 2021, pp. 1230–1235.

[21] Y. Li, H. Zhang, R. Li, J. Wang, W. Tong, G. Yan, and Z. Ma, "The complete affine automorphism group of polar codes," in *IEEE Global Communications Conference (GLOBECOM)*. December, 2021, pp. 01–06.

[22] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*. Elsevier, 1977, vol. 16.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[24] P. Trifonov and V. Miloslavskaya, "Polar subcodes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 254–266, 2015.

[25] E. Arıkan, "From sequential decoding to channel polarization and back again," *arXiv preprint arXiv:1908.09594*, 2019.

[26] "5G; NR; Multiplexing and channel coding (3GPP TS 38.212 version 17.4.0 Release 17)," 3rd Generation Partnership Project (3GPP), Tech. Rep., 2023.

[27] H. Yao, A. Fazeli, and A. Vardy, "List decoding of Arıkan's PAC codes," *Entropy*, vol. 23, no. 7, p. 841, 2021.

[28] M. Rowshan, A. Burg, and E. Viterbo, "Polarization-adjusted convolutional (PAC) codes: Sequential decoding vs list decoding," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1434–1447, 2021.

[29] G. C. Clark Jr and J. B. Cain, *Error-correction coding for digital communications*. Springer Science & Business Media, 2013.

[30] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On complexity of trellis structure of linear block codes," *IEEE Transactions on information theory*, vol. 39, no. 3, pp. 1057–1064, 1993.

[31] R. H. Morelos-Zaragoza, *The art of error correcting coding*. John Wiley & Sons, 2006.

[32] C. Schürch, "A partial order for the synthesized channels of a polar code," in *Proceedings of IEEE International Symposium on Information Theory*. July, 2016, pp. 220–224.

Hanwen Yao (Member, IEEE) received his B.S. degree in electrical engineering from the Shanghai Jiao Tong University in Shanghai, China, in 2016, and his M.S. and Ph.D. degrees in electrical engineering from the University of California, San Diego (UCSD) in 2018 and 2022, respectively. He received the Jack Keil Wolf ISIT Student Paper Award in 2021. He was also awarded the Shannon Graduate Fellowship by the Center for Memory and Recording Research in UCSD in 2021. He is currently a Postdoctoral Associate in the Department of Electrical and Computer Engineering with Duke University. His research interests include coding theory, combinatorics, and algorithm.

Arman Fazeli (Member, IEEE) received the B.S. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2012, and the Ph.D. degree in electrical and computer engineering from the University of California at San Diego. He was a Post-Doctoral Scholar and a Lecturer at the University of California at San Diego until 2021. He is currently a Wireless System Engineer at Apple. His main research interests include coding and information theory, with emphasis on polar codes and coding for distributed storage systems. He received Silver and Bronze Medals at the International Mathematical Olympiad (IMO) in 2006 and 2007, when he was in the Iran National Mathematics Team.

Alexander Vardy (Fellow, IEEE) was born in Moscow, Russia, in 1963. He received the B.Sc. degree (*summa cum laude*) from the Technion—Israel Institute of Technology, Israel, in 1985, and the Ph.D. degree from Tel-Aviv University, Israel, in 1991.

From 1985 to 1990, he was with the Israeli Air Force, where he worked on electronic counter measures systems and algorithms. From 1992 to 1993, he was a Visiting Scientist with the IBM Almaden Research Center, San Jose, CA, USA. From 1993 to 1998, he was with the University of Illinois at Urbana-Champaign, first as an Assistant Professor and then as an Associate Professor. Since 1998, he has been with the University of California at San Diego (UCSD), where he is currently the Jack Keil Wolf Chair Professor with the Department of Electrical and Computer Engineering and the Department of Computer Science. While on sabbatical from UCSD, he has held long-term visiting appointments with CNRS, France, the EPFL, Switzerland, the Technion—Israel Institute of Technology, and Nanyang Technological University, Singapore. His research interests include error-correcting codes, algebraic and iterative decoding algorithms, lattices and sphere packings, coding for storage systems, cryptography, computational complexity theory, and fun math problems. He was a member of the Board of Governors of the IEEE Information Theory Society from 1998 to 2006 and from 2011 to 2017. In 1996, he became a fellow of the David and Lucile Packard Foundation. He received the IBM Invention Achievement Award in 1993 and the NSF Research Initiation and CAREER Awards in 1994 and 1995. In 1996, he was appointed as a fellow of the Center for Advanced Study, University of Illinois, and received the Xerox Award for Faculty Research. He received the IEEE Information Theory Society Paper Award (jointly with Ralf Koetter) in 2004. In 2005, he received the Fulbright Senior Scholar Fellowship and the Best Paper Award at the IEEE Symposium on Foundations of Computer Science (FOCS). In 2017, his work on polar codes was recognized by the IEEE Communications and Information Theory Societies Joint Paper Award. From 1995 to 1998, he was an Associate Editor for *Coding Theory*. From 1998 to 2001, he was the Editor-in-Chief of the IEEE TRANSACTIONS ON INFORMATION THEORY.