

# List Decoding of Arikan's PAC Codes

Hanwen Yao, Arman Fazeli, and Alexander Vardy

## Abstract

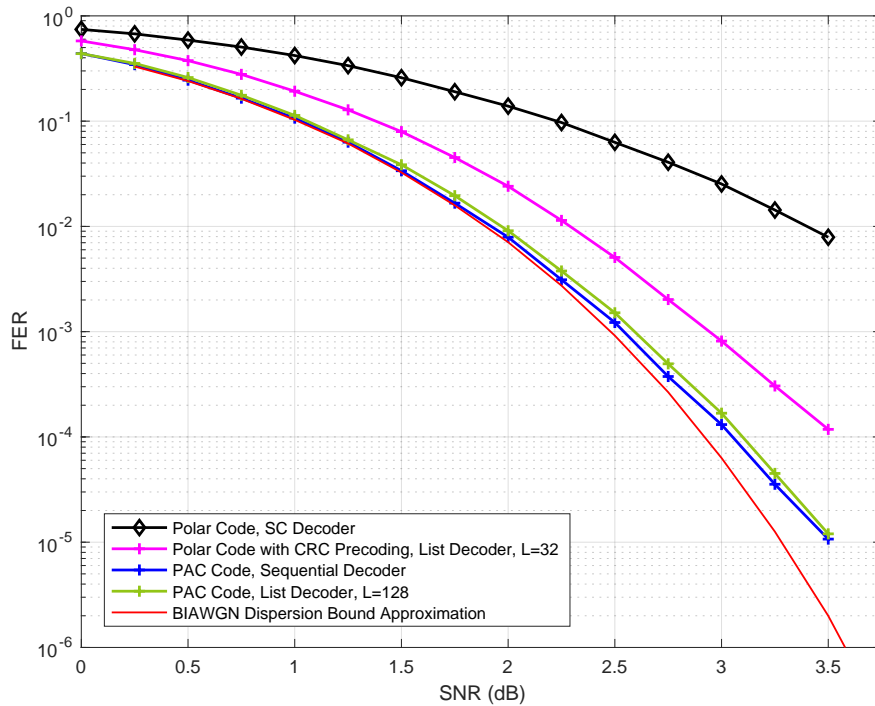
Polar coding gives rise to the first explicit family of codes that provably achieve capacity with efficient encoding and decoding for a wide range of channels. Recently, Arikan presented a new polar coding scheme, which he called *polarization-adjusted convolutional (PAC) codes*. At short blocklengths, such codes offer a dramatic improvement in performance as compared to CRC-aided list decoding of conventional polar codes. PAC codes are based primarily upon the following main ideas: replacing CRC codes with convolutional precoding (under appropriate rate profiling) and replacing list decoding by *sequential decoding*. Simulation results show that PAC codes, resulting from the combination of these ideas, are close to finite-length lower bounds on the performance of *any* code under ML decoding.

One of our main goals in this paper is to answer the following question: is sequential decoding essential for the superior performance of PAC codes? We show that similar performance can be achieved using *list decoding* when the list size  $L$  is moderately large (say,  $L \geq 128$ ). List decoding has distinct advantages over sequential decoding in certain scenarios, such as low-SNR regimes or situations where the worst-case complexity/latency is the primary constraint. Another objective is to provide some insights into the remarkable performance of PAC codes. We first observe that both sequential decoding and list decoding of PAC codes closely match ML decoding thereof. We then estimate the number of low weight codewords in PAC codes, and use these estimates to approximate the union bound on their performance. These results indicate that PAC codes are superior to both polar codes and Reed-Muller codes. We also consider random time-varying convolutional precoding for PAC codes, and observe that this scheme achieves the same superior performance with constraint length as low as  $\nu = 2$ .

## I. INTRODUCTION

Polar coding, pioneered by Arikan [2], gives rise to the first explicit family of codes that provably achieve capacity for a wide range of channels with efficient encoding and decoding. However, it is well known that at short block lengths the performance of polar codes is far from optimal.

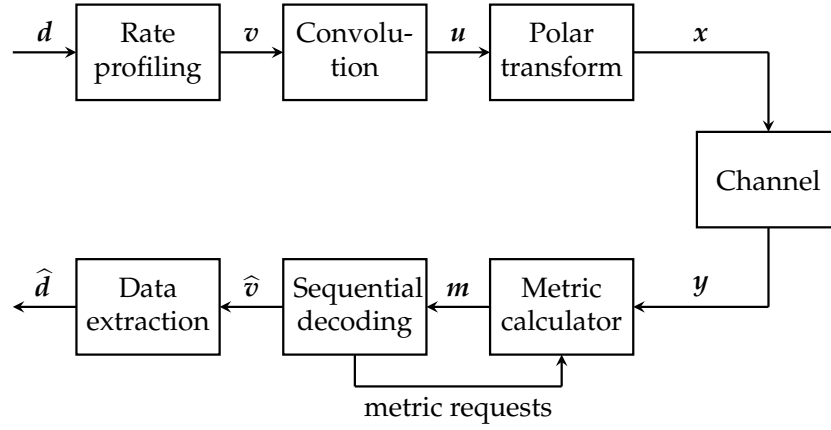
The authors are with the Department of Electrical and Computer Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA (emails: hwyao@ucsd.edu, afazelic@ucsd.edu, avardy@ucsd.edu). This work was presented in part at the *IEEE International Symposium on Information Theory* in June 2020.



**Figure 1.** Performance of PAC codes versus polar codes

For example, the performance of a polar code of length 128 and rate  $1/2$  on the binary-input AWGN channel under standard successive cancellation (SC) decoding is shown in Figure 1. Figure 1 largely reproduces the simulation results presented by Arikan in [3]. Codes of length 128 and rate  $1/2$  serve as the running example throughout Arikan's recent paper [3], and we will also adopt this strategy herein. We make no attempt to optimize these codes; rather, our goal is to follow Arikan [3] as closely as possible. Also shown in Figure 1 is the BIAWGN dispersion bound approximation for such codes. This can be thought of as an estimate of the performance of random codes under ML decoding (see [33]). Clearly, at length 128, there is a tremendous gap between polar codes under SC decoding and the best achievable performance.

As shown in [40] and other papers, the reasons for this gap are two-fold: the polar code itself is weak at such short lengths and SC decoding is weak in comparison with ML decoding. A well-known way to address both problems is CRC precoding followed by successive-cancellation list (SCL) decoding. Following [3], the performance of CRC-aided polar codes (with 8-bit CRC) of rate  $1/2$  under SCL decoding with list-size 32 is also shown in Figure 1. This approach, along with various refinements thereof (see [19], [22], [32] and other papers), has largely remained the state of the art in polar coding since it was first introduced in [40]. It is currently used as the coding scheme for control and physical broadcast channels in the enhanced mobile broadband



**Figure 2.** PAC coding scheme, reproduced from [3]

(eMBB) mode and the ultra-reliable low latency communications (URLLC) mode of the fifth generation (5G) wireless communications standard [59], [60].

In his Shannon Lecture at the ISIT in 2019, Erdal Arkan presented a significant breakthrough in polar coding, which significantly boosts the performance of polar codes at short lengths. Specifically, Arkan [3] proposed a new polar coding scheme, which he calls *polarization-adjusted convolutional (PAC) codes*. Remarkably, under sequential decoding, the performance of PAC codes is very close to the BIAWGN dispersion bound approximation [8], [33]. The performance of PAC codes of length 128 and rate  $1/2$  is also shown (in blue and green) in Figure 1.

#### A. Brief Overview of PAC Codes

Arkan's PAC codes [3] are largely based upon the following two innovations: replacing CRC precoding with *convolutional precoding* (under appropriate rate-profiling, discussed later in this section) and replacing list decoding by *sequential decoding*. The encoding and decoding of PAC codes are shown schematically in Figure 2, which is reproduced from [3].

Referring to Figure 2, let us consider an  $(n, k)$  PAC code. On the encoding side, Arkan uses a rate-1 convolutional precoder concatenated with a standard polar encoder. Only  $k$  out of the  $n$  bits of the input  $v$  to the convolutional precoder carry the information (or data) vector  $d$ . The remaining  $n - k$  bits of  $v$  are set to 0. Just like for conventional polar codes, the overall performance crucially depends upon *which positions* in  $v$  carry information and which are frozen to 0. This choice of frozen positions in  $v$ , Arkan has termed **rate-profiling**. Unlike conventional polar

codes, the optimal rate-profiling choice is *not* known. In fact, it is not even clear what optimization criterion should govern this choice, although we hope to shed some light on this in Section V.

The main operation on the decoder side is sequential decoding. Specifically, Arikan employs Fano decoding (as described in [9] and [16, Section 6.9]) of the convolutional code to estimate its input  $v$ . The path metrics used by this sequential decoder are obtained via repeated calls to the successive-cancellation decoder for the underlying polar code.

### B. Our Contributions

One of our main goals in this paper is to answer the following question: is sequential decoding *essential* for the superior performance of PAC codes? Is it possible, or perhaps advantageous, to replace the sequential decoder in Figure 2 by an alternative decoding method? We show that, indeed, similar performance can be achieved using list decoding, provided the list size  $L$  is moderately large. This conclusion is illustrated in Figure 1, where we use a list of size  $L = 128$  to closely match the performance of the sequential decoder. It remains to be seen which of the two approaches is advantageous in terms of complexity. While a comprehensive answer to this question would require implementation in hardware, we carry out a qualitative complexity comparison in Section IV. This comparison indicates that list decoding has distinct advantages over sequential decoding in certain scenarios. In particular, list decoding is certainly advantageous in low-SNR regimes or in situations where the *worst-case* complexity/latency is the primary constraint.

Another objective of this paper is to provide some insights into the remarkable performance of PAC codes observed in simulations. Although theoretical analysis of list decoding remains an open problem even for conventional polar codes, it has been observed in numerous studies that list decoding quickly approaches the performance of maximum-likelihood decoding with increasing list size  $L$ . As expected, we find this to be the case for PAC codes as well (see Figure 7). Fortunately, maximum-likelihood decoding of linear codes is reasonably well understood: its performance is governed by their weight distribution, and can be well approximated by the union bound, especially at high SNRs. Motivated by this observation, we use the method of [19] to estimate the number of low-weight codewords in PAC codes, under polar and RM rate profiles (introduced by Arikan [3]). We find that PAC codes with the RM rate-profile are superior to both polar codes (with or without CRC precoding) *and* the  $(128, 64, 16)$  Reed-Muller code. For more on this, see Table 1 and Figure 9. We also introduce and study random time-varying convo-

lutional precoding for PAC codes. We find that, as compared with the convolutional precoding introduced in [3], time-varying convolutional precoding is much less sensitive to the constraint length. Arikan uses in [3] a convolutional code generated by  $c = (1, 0, 1, 1, 0, 1, 1)$ , whose constraint length is  $\nu = 6$ . In Figure 12, we observe that under list decoding, random time-varying precoding achieves essentially the same performance with constraint length  $\nu = 2$ .

### C. Related Work

Numerous attempts have been made to improve the performance of polar codes at short block lengths. Various approaches based on replacing successive-cancellation decoding with more advanced decoders include list decoding [40], adaptive list decoding [19], sequential decoding [22], [43], and stack decoding [32], among others. When concatenating a polar code with an outer code, most of the existing work still uses CRC outer codes and their variants, as originally proposed in [40]. However, many other modifications of the basic polar-coding paradigm have been extensively studied, including large polarization kernels [10], [12], [18], [31], [44], [45], [50], [51], [53], polar subcodes [29], [42], [44], [46]–[49], “convolutional” polar codes [14], [15], [28], [30], and polarized Reed-Muller coding [1], [20], [25], [56] among others.

As shown later in this paper, in Arikan’s PAC codes, convolutional precoding combined with rate-profiling can be regarded as replacing traditional frozen bits with *dynamically frozen bits*. Polar coding with dynamically frozen bits was first introduced by Trifonov and Miloslavskaya in [47], and later studied in [7], [23], [29], [42], [44], [47]–[49], [57] and other papers. However, the dynamic freezing patterns in these papers are very different from [3]. Prior to Arikan’s work [3], *convolutional precoding of polar codes* was proposed in [11] and later studied in [13].

Although quite recent, Arikan’s PAC codes have already attracted considerable interest; see for example [4], [21], [24], [26], [27], [41], [52]. While these papers investigate various aspects of PAC codes, none of them considers list decoding thereof. Finally, we note the work of [34] and [35], which investigates both Fano decoding and list decoding of PAC codes. This work is apparently independent from and contemporaneous with our results herein<sup>1</sup>.

<sup>1</sup>The paper of Rowshan, Burg, and Viterbo [34] was posted on [arxiv.org](https://arxiv.org) in February 2020, while our work [54] was submitted for review in January 2020. Our results became available on [arxiv.org](https://arxiv.org) in May 2020. The Rowshan-Viterbo paper [35] was posted on [arxiv.org](https://arxiv.org) in July 2020, after our results were presented in [54].

### D. Paper Outline

The rest of this paper is organized as follows. We begin with an overview on Arkan's PAC codes in Section II, including both their encoding process and sequential decoding. In Section III, we present our list-decoding algorithm. In Section IV, we compare it with sequential decoding, in terms of both performance and complexity. In Section V, we endeavor to acquire some insight into the remarkable performance of PAC codes. First, we show empirically that both sequential decoding and list decoding thereof are extremely close to the ML decoding performance. To get a handle on the latter, we estimate the number of low-weight codewords in PAC codes (and polar codes) under different rate profiles. This makes it possible to approximate the performance of ML decoding with a union bound. In Section VI, we introduce and study random time-varying convolutional precoding for PAC codes, and show that it may be advantageous in terms of the constraint length. We conclude with a brief discussion in Section VII.

## II. OVERVIEW OF ARIKAN'S PAC CODES

For details on conventional polar codes under standard SC decoding, we refer the reader to Arkan's seminal paper [2]. Like polar codes, the block length  $n$  of a PAC code is also a power of 2. That is,  $n = 2^m$  with  $m \geq 1$ . As shown in Figure 2, the encoding process for an  $(n, k)$  PAC code consists of the following three steps: rate-profiling, convolutional precoding, and polar encoding. In the first step, the  $k$  data (information) bits of the data vector  $\mathbf{d}$  are embedded into a data-carrier vector  $\mathbf{v}$  of length  $n$ , at  $k$  positions specified by an index set  $\mathcal{A} \subseteq \{0, 1, \dots, n-1\}$  with  $|\mathcal{A}| = k$ . The remaining  $n - k$  positions in  $\mathbf{v}$  are frozen to zero. Arkan [3] used **rate-profiling** to refer to this step, along with the choice of the index set  $\mathcal{A}$ .

Just like for polar codes, a careful choice of the index set  $\mathcal{A}$  is crucial to achieve good performance. Arkan has proposed in [3] two alternative approaches for selecting this set  $\mathcal{A}$ . The first approach, called **polar rate-profiling**, proceeds as follows. Let  $W_0, W_1, \dots, W_{n-1}$  be the  $n$  bit-channels, defined with respect to the conventional polar code of length  $n$ . In polar rate-profiling,  $\mathcal{A}$  is chosen so that  $\{W_i : i \in \mathcal{A}\}$  consists of the  $k$  best bit-channels in terms of their capacity. In other words, the capacities of the  $k$  bit-channels  $\{W_i : i \in \mathcal{A}\}$  are the  $k$  highest values among  $I(W_0), I(W_1), \dots, I(W_{n-1})$ . The second approach proposed in [3] is called **RM rate-profiling**. Let  $\text{wt}(i)$  denote the Hamming weight of the binary expansion of an index  $i$ . In RM rate-profiling, we simply pick the  $k$  indices of the highest weight, with ties

resolved arbitrarily. In other words, the set  $\{\text{wt}(i) : i \in \mathcal{A}\}$  consists of the  $k$  largest values among  $\text{wt}(0), \text{wt}(1), \dots, \text{wt}(n-1)$ . Notably, *without convolutional precoding*, this choice of  $\mathcal{A}$  generates Reed-Muller codes (as subcodes of a rate-1 polar code).

In the second step, the data-carrier vector  $\mathbf{v}$  resulting from the rate-profiling step is encoded using a rate-1 convolutional code generated by  $\mathbf{c} = (c_0, c_1, \dots, c_\nu)$ , with  $c_0 = c_\nu = 1$  (the latter can be assumed without loss of generality). This produces another vector  $\mathbf{u} = (u_0, u_1, \dots, u_{n-1})$  of length  $n$ , where

$$u_0 = c_0 v_0, \quad u_1 = c_0 v_1 + c_1 v_0, \quad u_2 = c_0 v_2 + c_1 v_1 + c_2 v_0,$$

and so on. In general, every bit in  $\mathbf{u}$  is a linear combination of  $(\nu + 1)$  bits of  $\mathbf{v}$  computed via the convolution operation:

$$u_i = \sum_{j=0}^{\nu} c_j v_{i-j} \quad (1)$$

where for  $i - j < 0$ , we set  $v_{i-j} = 0$  by convention. Alternatively, this step can be viewed as a vector-matrix multiplication  $\mathbf{u} = \mathbf{v}\mathbf{T}$ , where  $\mathbf{T}$  is the upper-triangular Toeplitz matrix:

$$\mathbf{T} = \begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_\nu & 0 & \cdots & 0 \\ 0 & c_0 & c_1 & c_2 & \cdots & c_\nu & & \vdots \\ 0 & 0 & c_0 & c_1 & \ddots & \cdots & c_\nu & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & & & \ddots & 0 & c_0 & c_1 & c_2 \\ \vdots & & & & 0 & 0 & c_0 & c_1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & c_0 \end{bmatrix} \quad (2)$$

In the third step, the vector  $\mathbf{u}$  is finally encoded by a conventional polar encoder as the codeword  $\mathbf{x} = \mathbf{u}\mathbf{P}_m$ . Here

$$\mathbf{P}_m = \mathbf{B}_n \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes m}$$

where  $\mathbf{B}_n$  is the  $n \times n$  bit-reversal permutation matrix (as defined in [2, Section VII]), and  $\mathbf{P}_m$  is known as the *polar transform matrix*.

With reference to the foregoing discussion, the PAC code in Figure 1 is obtained via RM rate-profiling using the rate-1 convolutional code generated by  $\mathbf{c} = (1, 0, 1, 1, 0, 1, 1)$ . This produces the (128, 64) PAC code of rate  $1/2$ , which is the code studied by Arikan in [3]. This specific PAC code will serve as our primary running example throughout the paper.

On the decoding side, Arikan [3] employs sequential decoding of the underlying convolutional code to decode the data-carrier vector  $\mathbf{v}$ . Under the frozen-bit constraints imposed by rate-profiling, the rate-1 convolutional code becomes an irregular tree code. There are many different

variants of sequential decoding for irregular tree codes, varying in terms of both the decoding metric used and the algorithm itself. Arikan [3] uses the *Fano sequential decoder*, described in [9] and [16]. Notably, the path metrics at the input to the sequential decoder are obtained via repeated calls to the successive-cancellation decoder for the underlying polar code, as shown in Figure 2.

### III. LIST DECODING OF PAC CODES

One of our main objectives herein is to determine whether sequential decoding of PAC codes (cf. Figure 2) can be replaced by list decoding. In this section, we show how list decoding of PAC codes can be implemented efficiently. In the next section, we will consider the performance and complexity of the resulting decoder, as compared to the sequential decoder of [3].

#### A. PAC Codes as Polar Codes with Dynamically Frozen Bits

To achieve efficient list decoding of PAC codes, we use the list-decoding algorithm developed in [40]. The complexity of this algorithm is  $O(Ln \log n)$ , where  $L$  is the list size. However, the algorithm of [40] decodes *conventional polar codes*. In order to make it possible to decode PAC codes with (a modified version of) this algorithm, we first observe that PAC codes can be regarded as *polar codes with dynamically frozen bits*.

Polar coding with dynamically frozen bits was first introduced by Trifonov and Miloslavskaya in [47], and later studied by the same authors in [48] and [49]. Let us briefly describe the general idea. In conventional polar coding, it is common practice to set all frozen bits to zero. That is,  $u_i = 0$  for all  $i \in \mathcal{F}$ , where  $\mathcal{F} \subset \{0, 1, \dots, n-1\}$  denotes the set of frozen indices. However, this choice is arbitrary: we can set  $u_i = 1$  for some  $i \in \mathcal{F}$  and  $u_i = 0$  for other  $i \in \mathcal{F}$ . What matters is that the frozen bits are fixed and, therefore, known *a priori* to the decoder. In [47], it was further observed that in order to be known *a priori* to the decoder, the frozen bits do *not* have to be fixed. Given  $i \in \mathcal{F}$ , we can set

$$u_i = f_i(u_0, u_1, \dots, u_{i-1}) \quad (3)$$

where  $f_i$  is a fixed Boolean function (usually, a linear function) known *a priori* to the decoder. For all  $i \in \mathcal{F}$ , the decoder can then decide as follows

$$\hat{u}_i = f_i(\hat{u}_0, \hat{u}_1, \dots, \hat{u}_{i-1}) \quad (4)$$

where  $\hat{u}_0, \hat{u}_1, \dots, \hat{u}_{i-1}$  are its earlier decisions. The encoding/decoding process in (3) and (4) is known as dynamic freezing.



In order to explain how Arıkan's PAC codes [3] fit into the dynamic freezing framework, let us first introduce some notation. With reference to Section II, for  $i = 0, 1, \dots, n-1$ , let  $\mathbf{u}_i$  and  $\mathbf{v}_i$  denote the vectors  $(u_0, u_1, \dots, u_i)$  and  $(v_0, v_1, \dots, v_i)$ , respectively. Further, let  $\mathbf{T}_{i,j}$  denote the submatrix of the Toeplitz matrix  $\mathbf{T}$  in (2), consisting of the first (topmost)  $i+1$  rows and the first (leftmost)  $j+1$  columns. With this, it is easy to see that  $\mathbf{u}_i = \mathbf{v}_i \mathbf{T}_{i,i}$  for all  $i$ . The matrix  $\mathbf{T}_{i,i}$  is upper triangular with  $\det \mathbf{T}_{i,i} = c_0 = 1$ . Hence it is invertible, and we have  $\mathbf{v}_i = \mathbf{u}_i \mathbf{T}_{i,i}^{-1}$  for all  $i$ . Now suppose that  $i \in \mathcal{A}^c$ , so that  $\mathbf{v}_i$  is frozen to zero in the rate-profiling step. Then we have

$$\mathbf{u}_i = \mathbf{v}_{i-1} \mathbf{T}_{i-1,i} = (\mathbf{u}_{i-1} \mathbf{T}_{i-1,i-1}^{-1}) \mathbf{T}_{i-1,i} \quad (5)$$

In particular, this means that the last bit  $u_i$  of the vector  $\mathbf{u}_i$  is an *a priori* fixed linear function of its first  $i$  bits, as follows:

$$u_i = (u_0, u_1, \dots, u_{i-1}) \mathbf{T}_{i-1,i-1}^{-1} (0, \dots, 0, c_\nu, c_{\nu-1}, \dots, c_1)^t$$

where  $(0, \dots, 0, c_\nu, c_{\nu-1}, \dots, c_1)^t$  represents the last column of the matrix  $\mathbf{T}_{i-1,i}$ . Clearly, the above is a special case of dynamic freezing in (3). Moreover, it follows that the set  $\mathcal{F}$  of indices that are dynamically frozen is precisely the same as in the rate-profiling step, that is  $\mathcal{F} = \mathcal{A}^c$ .

If  $i \in \mathcal{A}$ , then  $v_i$  is an information bit, but the value of  $u_i$  is determined not only by  $v_i$  but by  $v_{i-1}, v_{i-2}, \dots, v_{i-\nu}$  as well. Thus when representing PAC codes as polar codes, the information bits may be also regarded as dynamic.

Finally, note that in implementing the PAC decoder, there is no need to actually invert a matrix as in (5). Instead, we can successively compute the vector  $\hat{\mathbf{v}} = (\hat{v}_0, \hat{v}_1, \dots, \hat{v}_{n-1})$  as follows. If  $i \in \mathcal{A}^c$ , set  $\hat{v}_i = 0$ . Otherwise, set

$$\hat{v}_i = \hat{u}_i - \sum_{j=1}^{\nu} c_j \hat{v}_{i-j} \quad (6)$$

where the value of  $\hat{u}_i$  is provided by the polar decoder. Given  $\hat{v}_i, \hat{v}_{i-1}, \dots, \hat{v}_{i-\nu}$ , the values of the dynamically frozen bits  $\hat{u}_i$  for  $i \in \mathcal{A}^c$  can be computed using (1). This computation, along with the one in (6), takes linear time. All that is required is additional memory to store the vector  $\hat{\mathbf{v}} = (\hat{v}_0, \hat{v}_1, \dots, \hat{v}_{n-1})$ .

### B. List Decoding of PAC Codes

Representing PAC codes as polar codes with dynamically frozen bits makes it possible to adapt existing algorithms for successive-cancellation list decoding of polar codes to decode PAC codes.

**Algorithm 1:** List Decoder for PAC Codes

**Input:** The received vector  $\mathbf{y}$ , the list size  $L$ , the generator  $\mathbf{c} = (c_0, c_1, \dots, c_v)$  for the convolutional precoder as global

**Output:** Decoded codeword  $\hat{\mathbf{x}}$

```

// Initialization
1 ... lines 2–5 of Algorithm 12 in [40]
2 shiftRegisters  $\leftarrow$  new 2-D array of size  $L \times (v + 1)$ 
3 for  $\ell = 0, 1, \dots, L - 1$  do
4   shiftRegister[ $\ell$ ] =  $(0, 0, \dots, 0)$ 

// Main Loop
5 for  $\phi = 0, 1, \dots, n - 1$  do
6   recursivelyCalcP( $m, \phi$ )
7   if  $u_\phi$  is frozen then
8     for  $\ell = 0, 1, \dots, L - 1$  do
9       if activePath[ $\ell$ ] = false then
10        continue
11      left-shift shiftRegister[ $\ell$ ] by one, with the rightmost position set to 0
12       $C_m \leftarrow$  getArrayPointerC( $m, \ell$ )
13       $(v_{\phi-v}, v_{\phi-v+1}, \dots, v_\phi) \leftarrow$  shiftRegister[ $\ell$ ]
14      // Set the frozen bit
15       $C_m[0][\phi \bmod 2] \leftarrow \sum_{j=0}^v c_j v_{\phi-j}$ 
16   else
17     continuePaths_Unfzn( $\phi$ )
18   if  $\phi \bmod 2 = 1$  then
19     recursivelyUpdateC( $m, \phi$ )

// Get the best codeword in the list
19 ... lines 17–24 of Algorithm 12 in [40]
20 return  $\hat{\mathbf{x}} = (C_0[\beta][0])_{\beta=0}^{n-1}$ 

```

**Algorithm 2:** continuePaths\_Unfzn (PAC version)**Input:** phase  $\phi$ 


---

```

1 ... lines 1–18 of Algorithm 13 in [40]
  // Continue relevant paths
2 for  $\ell = 0, 1, \dots, L - 1$  do
3   if  $\text{contForks}[\ell][0] = \text{false}$  and  $\text{contForks}[\ell][1] = \text{false}$  then
4     continue
5    $C_m \leftarrow \text{getArrayPointer\_C}(m, \ell)$ 
6   left-shift  $\text{shiftRegister}[\ell]$  by one, with the rightmost position set to 0
7    $(v_{\phi-v}, v_{\phi-v+1}, \dots, v_{\phi}) \leftarrow \text{shiftRegister}[\ell]$ 
8   if  $\text{contForks}[\ell][0] = \text{true}$  and  $\text{contForks}[\ell][1] = \text{true}$  then
9      $C_m[0][\phi \bmod 2] \leftarrow \sum_{j=0}^v c_j v_{\phi-j}$ 
10     $\ell' \leftarrow \text{clonePath}(\ell)$ 
11     $\text{shiftRegister}[\ell'] \leftarrow \text{shiftRegister}[\ell]$ 
12    flip the rightmost bit of  $\text{shiftRegister}[\ell']$ 
13     $C_m \leftarrow \text{getArrayPointer\_C}(m, \ell')$ 
14     $(v'_{\phi-v}, v'_{\phi-v+1}, \dots, v'_{\phi}) \leftarrow \text{shiftRegister}[\ell']$ 
15     $C_m[0][\phi \bmod 2] \leftarrow \sum_{j=0}^v c_j v'_{\phi-j}$ 
16  else
17    if  $\text{contForks}[\ell][0] = \text{true}$  then
18      if  $\sum_{j=0}^v c_j v_{\phi-j} = 1$  then
19        flip the rightmost bit of  $\text{shiftRegister}[\ell]$ 
20        set  $C_m[0][\phi \bmod 2] \leftarrow 0$ 
21    else
22      if  $\sum_{j=0}^v c_j v_{\phi-j} = 0$  then
23        flip the rightmost bit of  $\text{shiftRegister}[\ell]$ 
24        set  $C_m[0][\phi \bmod 2] \leftarrow 1$ 

```

---

There are, however, a few important differences. For example, for conventional polar codes, when the list decoder encounters a frozen index  $i \in \mathcal{F}$ , all the paths in the list-decoding tree are extended in the same way, by setting  $\hat{u}_i = 0$ . For PAC codes, since freezing is dynamic, different paths are potentially extended differently, depending upon the previous decisions along the path.

In general, our list decoder for PAC codes maintains the same data structure as the successive-cancellation list decoder in [40]. In addition, for a list of size  $L$ , we introduce  $L$  auxiliary shift registers — one for each path. Each such shift register stores the last  $\nu$  bits of the vector  $\hat{\mathbf{v}} = (\hat{v}_0, \hat{v}_1, \dots, \hat{v}_{n-1})$ , computed as in (6), for the corresponding path.

Algorithm 1 and Algorithm 2 provide the full details of our list decoding algorithm for PAC codes. These algorithms fit into the same general mold as Algorithms 12 and 13 of [40], with the differences highlighted in blue.

#### IV. LIST DECODING VERSUS SEQUENTIAL DECODING

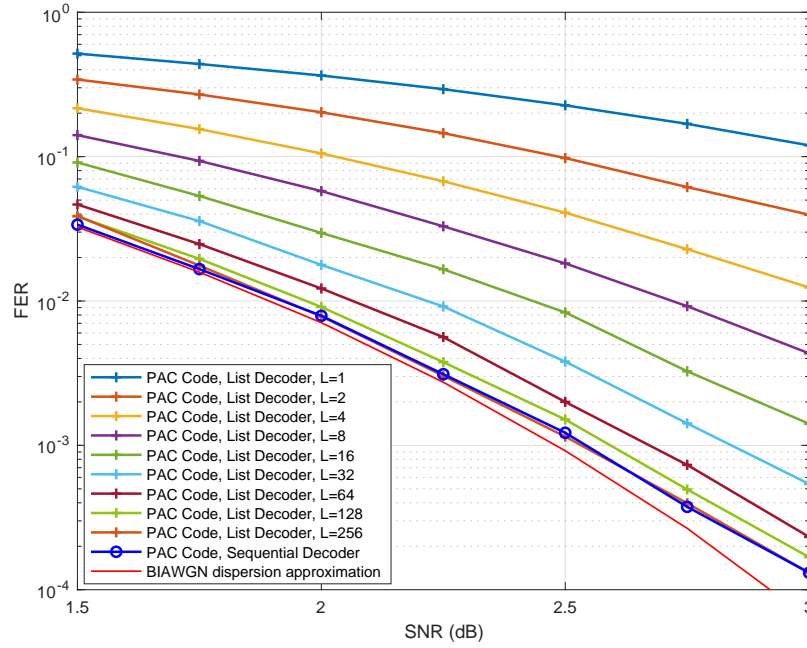
We now compare list decoding of PAC codes with sequential decoding, in terms of both performance and complexity. For list decoding, we use the algorithm of Section III. For sequential decoding, we employ exactly the same Fano decoder that was used by Arikan in [3]. We are grateful to Erdal Arikan for sharing the details of his decoding algorithm. We do not disclose these details here, instead referring the reader to [3], [26], [27].

We note that more efficient algorithms for sequential decoding of polar codes and their sub-codes may be available; see in particular the work of Trifonov [43], [49]. However, in this paper, we use the results of Arikan [3] as a benchmark, in terms of both performance and complexity.

Our main conclusion is that sequential decoding is *not* essential in order to achieve the remarkable performance of PAC codes: similar performance can be obtained with list decoding, providing the list size is sufficiently large. As far as complexity, sequential decoding is generally better at high SNRs and in terms of average complexity, while list decoding is advantageous in terms of worst-case complexity and at low SNRs.

##### A. Performance Comparison

Figure 3 summarizes simulation results comparing the performance of the Fano decoder from [3] with our list decoding algorithm, as a function of the list size  $L$ . The underlying PAC code is the



**Figure 3.** Performance of PAC codes under list decoding

same as in Figure 1; it is the  $(128, 64)$  PAC code obtained via RM rate-profiling (see Section II). The underlying channel is the binary-input additive white Gaussian noise (BIAWGN) channel.

As expected, the performance of list decoding steadily improves with increasing list size. For  $L = 128$ , the list-decoding performance is very close to that of sequential decoding, while for  $L = 256$  the two curves virtually coincide over the entire range of SNRs.

It should be pointed out that the frame error rate (FER) reported for sequential decoding in Figures 1 and 3 is due to two different mechanisms of error/failure. In some cases, the sequential decoder reaches the end of the search tree (see Figure 4) producing an incorrect codeword. These are decoding errors. In other cases, the end of the search tree is never reached; instead, the computation is aborted once it exceeds a predetermined cap on the number of cycles. These are decoding failures. As in [3], the FER plotted in Figure 3 counts all the cases wherein the transmitted codeword is not produced by the decoder: thus it is the *sum of the error rate and the failure rate*. The table below shows what fraction of such cases were due to decoding failures:

SNR [dB]	1.00	1.25	1.50	1.75	2.00	2.25
% of failures	4.53%	3.56%	1.86%	1.38%	1.01%	0.29%

A decoding failure was declared in our simulations whenever the number of cycles (loosely speaking, cycles count forward and backward movements along the search tree in the Fano

decoder) exceeded 1,300,000. This is exactly the same cap on the number of cycles that was used by Arıkan in [3]. Overall, the foregoing table indicates that increasing this cap would not improve the performance significantly.

The FER for list decoding is also due to two distinct error mechanisms. In some cases, the transmitted codeword is not among the  $L$  codewords generated by our decoding algorithm. In other cases, it *is* on the list of codewords generated, but it is not the most likely among them. Since the list decoder selects the most likely codeword on the list as its ultimate output, this leads to a decoding error. We refer to such instances as selection errors. The table below shows the fraction of selection errors for lists of various sizes:

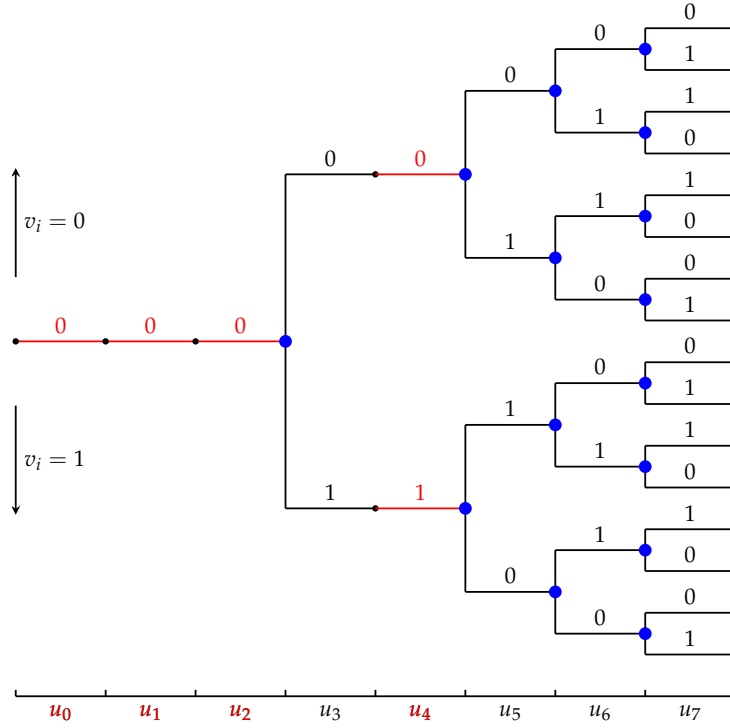
SNR [dB]	1.50	1.75	2.00	2.25	2.50	2.75	3.00
$L = 64$	32.1%	32.2%	32.5%	32.3%	29.4%	36.7%	39.6%
$L = 128$	50.0%	51.6%	54.6%	53.6%	58.4%	60.4%	63.2%
$L = 256$	66.2%	71.0%	75.2%	78.0%	79.9%	83.6%	82.8%

This indicates that the performance of list decoding would further improve (at least, for  $L \geq 64$ ) if we could somehow increase the minimum distance of the underlying code, or otherwise aid the decoder in selecting from the list (e.g. with CRC).

Finally, we also include in Figures 1 and 3 the BIAWGN dispersion-bound approximation for binary codes of rate  $1/2$  and length 128. The specific curve plotted in Figures 1 and 3 is the so-called *normal approximation* of the dispersion bound of Polyanskiy, Poor, and Verdu [33]. Our curve coincides with those given in [6, Figure 1] and [17, Figure 6]. Note that a more accurate bound can be derived using the methods of Erseghe [8], but this is not critical for our purposes. It is clear from Figures 1 and 3 that the performance of the  $(128, 64)$  PAC code, under both sequential decoding and list decoding with  $L \geq 128$ , is close to the best achievable performance.

### B. Complexity Comparison

A comprehensive complexity analysis of list decoding versus sequential decoding of PAC codes in practical applications is likely to require algorithmic optimization and implementation in hardware. In the case of list decoding, this should be relatively easy based upon our representation of PAC codes as polar codes with dynamically frozen bits (see Section III-A) in conjunction with existing work on efficient hardware implementation of polar list decoders (see [36], [37], for



**Figure 4.** An example of the polar search tree, reproduced from [3]

example). On the other hand, we are not aware of any existing implementations of sequential decoding in hardware. Such implementation may be challenging due to variable running time, which depends on the channel noise, and complex control logic [5].

In this section, we provide a qualitative comparison of list decoding versus sequential decoding using two generic complexity metrics: the number of nodes visited in the polar search tree and the total number of floating-point operations performed by the decoder. The results we obtain for the two metrics, summarized in Figures 5 and 6, are consistent with each other.

The polar search tree, shown schematically in Figure 4, represents all possible inputs  $\mathbf{u} = (u_0, u_1, \dots, u_{n-1})$  to the polar encoder. It is an irregular tree with  $n + 1$  levels containing  $2^k$  paths. If  $i \in \mathcal{A}^c$  then all nodes at level  $i$  have a single outgoing edge, as  $u_i$  is dynamically frozen in this case. In contrast with conventional polar codes, these edges may be labeled differently (cf.  $u_4$  in Figure 4). If  $i \in \mathcal{A}$  then all nodes at level  $i$  have two outgoing edges. In this framework, both list decoding and sequential decoding can be regarded as tree-search algorithms that try to identify the most likely path in the tree. The list decoder does so by following  $L$  paths in the tree, from the root to the leaves, and selecting the most likely one at the end. The Fano sequential decoder follows only one path, but has many back-and-forth movements during the decoding process.

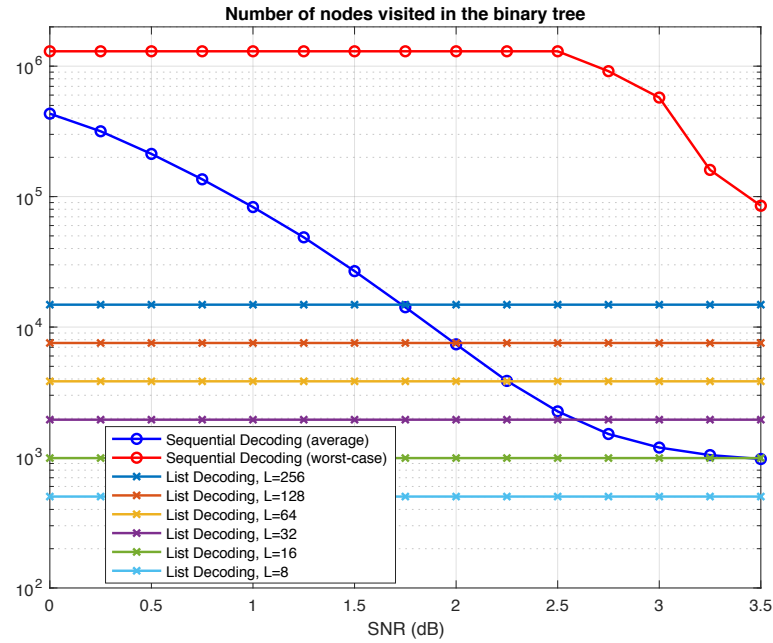


Figure 5. Sequential decoding vs. list decoding: Number of nodes visited in the polar search tree per codeword

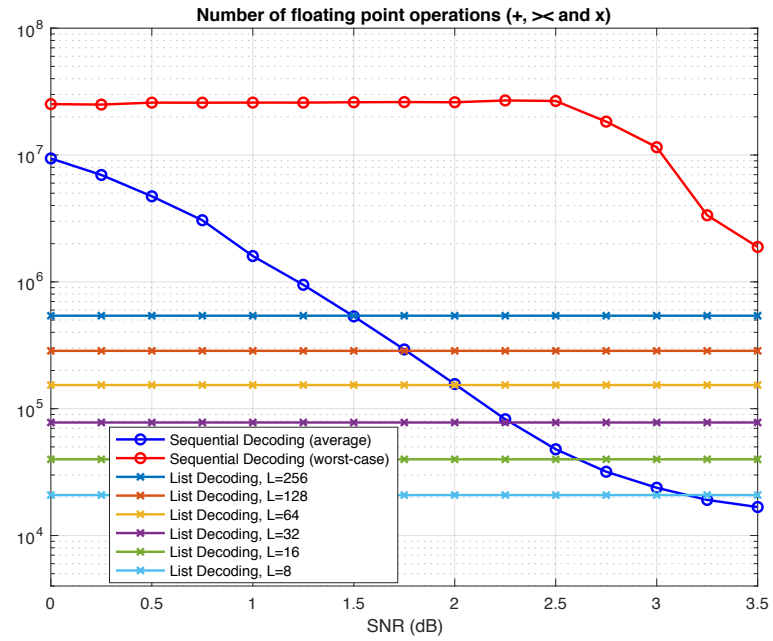


Figure 6. Sequential decoding vs. list decoding: Number of floating-point operations per decoded codeword

For the sake of qualitative comparison, we take the total number of nodes the two algorithms visit in the tree as one reasonable proxy of their complexity. In doing so, we disregard the nodes at the frozen levels, and count only those nodes that have two outgoing edges (colored blue in



Figure 4); we call them the *decision nodes*. Figure 5 shows the number of decision nodes visited by the two decoding algorithms as a function of SNR.

For sequential decoding, two phenomena are immediately apparent from Figure 5. First, there is a tremendous gap between worst-case complexity and average complexity. For most SNRs, the worst-case complexity is dominated by decoding failures, which trigger a computational timeout upon reaching the cap on the number of cycles (see Section IV-A). Clearly, reducing this cap would also reduce the worst-case complexity. On the other hand, for SNRs higher than 2.50 dB, decoding failures were not observed. Thus, beyond 2.50 dB, the worst-case complexity gradually decreases, as expected. Another phenomenon apparent from Figure 5 is that the average complexity is highly dependent on SNR. This is natural since the processing in the Fano sequential decoder depends on the channel noise. The less noise there is, the less likely is the sequential decoder to roll back in its search for a better path.

Neither of the two phenomena above is present for list decoding: the worst-case complexity is equal to the average complexity, and both are unaffected by SNR. The resulting curves in Figures 5 and 6 are flat, since the complexity of list decoding depends only on the list size  $L$  and the code dimension  $k$ .

In fact, the number of decision nodes visited by the list decoder in the polar search tree can be easily computed as follows. First assume, for simplicity, that  $L$  is a power of 2. As the list decoder proceeds from the root to the leaves, the number of paths it traces doubles for every  $i \in \mathcal{A}$  until it reaches  $L$ . The number of decision nodes it visits during this process is given by  $1 + 2 + 4 + \dots + L = 2L - 1$ . After reaching  $L$  paths, the decoder visits  $L$  decision nodes at every one of the remaining  $k - \log_2 L$  levels that are not frozen. Thus the total number of decision nodes visited is  $L(k + 2 - \log_2 L) - 1 = O(kL)$ . If  $L$  is not a power of 2, this counting argument readily generalizes, and the number of decision nodes visited is given by

$$L(k + 1 - \lceil \log_2 L \rceil) + 2^{\lceil \log_2 L \rceil} - 1 = O(kL) \quad (7)$$

As another qualitative metric of complexity of the two algorithms, we count the total number of additions, comparisons, and multiplications of floating-point numbers throughout the decoding process. The results of this comparison are compiled in Figure 6. The number of floating-point operations is a more precise measure of complexity than the number of decision nodes visited in the search tree. Yet we observe exactly the same pattern as in Figure 5. For list decoding, it is no longer possible to give a simple expression as in (7), but the complexity is still inde-

pendent of SNR, resulting in flat curves. For sequential decoding, we again observe the same two phenomena discussed earlier in connection with Figure 5. In particular, the worst-case complexity remains prohibitive even at high SNRs.

In summary, our qualitative comparison suggests that, for a similar level of performance, sequential decoding is clearly advantageous in terms of average-case complexity at high SNRs. However, list decoding may have distinct advantages in low-SNR regimes or in situations where the worst-case complexity/latency is the primary constraint.

## V. PERFORMANCE ANALYSIS FOR PAC CODES

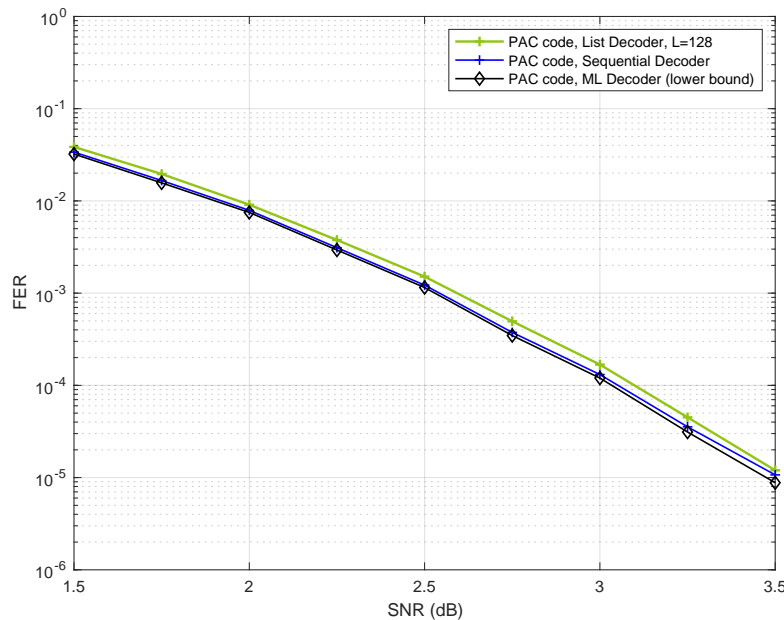
In this section, we study the performance of PAC codes under the assumption of maximum-likelihood (ML) decoding. To this end, we estimate computationally the number of low-weight codewords in PAC codes (and other codes), then combine these estimates with the union bound. First, we explain why analysis of performance under ML decoding makes sense in our setting.

### A. Sequential Decoding versus ML Decoding

It has been observed in several papers that for polar codes, list decoding rapidly approaches the performance of ML decoding with increasing list-size  $L$ . In this section, as expected, we find this to be the case for Arikan's (128,64) PAC code as well.

Figure 7 shows a bound on the frame error-rate of ML decoding obtained in our simulations. This is an empirical *lower bound*, in the sense that the actual simulated performance of ML decoding could only be worse — even closer to the other two curves (for sequential decoding and list decoding) shown in Figure 7. The bound was generated using the Fano sequential decoder, as follows. Every time the Fano decoder makes an error, we compare the likelihoods of the transmitted path and the path produced by the decoder. If the decoded path has a higher path-metric (likelihood), then the ML decoder will surely make an error in this instance as well. We count such instances to generate the lower bound. This method of estimating ML performance in simulations is very similar to the one introduced in [40] for polar codes, except that [40] used list decoding.

Figure 7 provides strong evidence that it makes sense to study PAC codes under ML decoding in order to gain insights into their performance under sequential decoding, since the two are remarkably close. Figure 7 also reveals one of the reasons why Arikan's PAC codes are so good at short blocklengths: they can be efficiently decoded with near-ML fidelity.



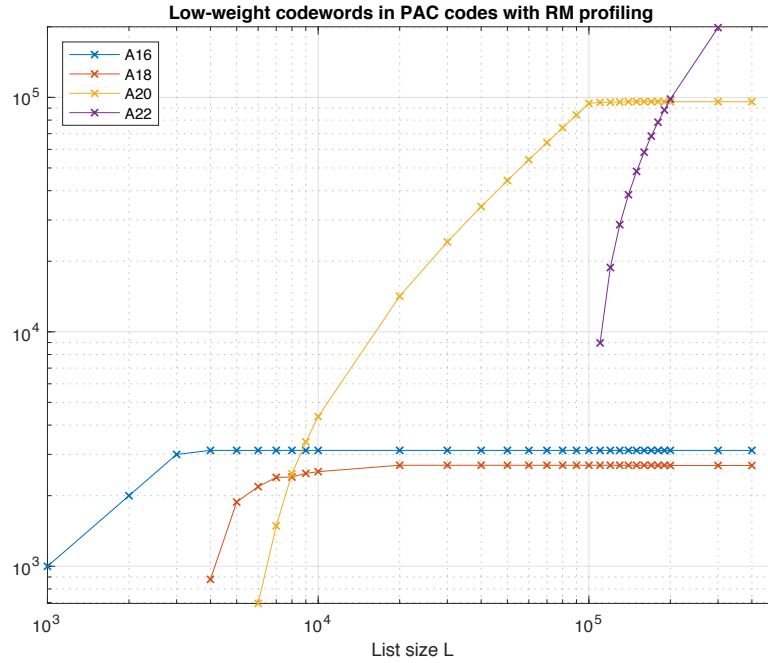
**Figure 7.** Performance of the PAC code under ML decoding

### B. Weight Distributions and Union Bounds

We now study the weight distribution of the  $(128, 64)$  PAC code in order to develop analytical understanding of its performance under ML decoding. Specifically, we use the method of [19] to estimate the number of low-weight codewords in this code.

Consider the following experiment, devised in [19]. Transmit the all-zero codeword in the extremely high SNR regime, and use list decoding to decode the channel output. It is reasonable to expect that in this situation, the list decoder will produce codewords of low weight. As  $L$  increases, since the decoder is forced to generate a list of size exactly  $L$ , more and more low-weight codewords emerge. The results of this experiment for the  $(128, 64)$  PAC code are shown in Figure 8 as a function of the list size. We can see that the only weights observed for  $L$  up to 400,000 are 16, 18, 20, and 22. Moreover,  $A_{16} \geq 3120$ ,  $A_{18} \geq 2696$ , and  $A_{20} \geq 95828$  (cf. Table 1). These numbers are *lower bounds* on the weight distribution of the code. However, the fact that the curves in Figure 8 *saturate* at these values provides strong evidence that these bounds are exact, and that codewords of other low weights do not exist.

We have used the same method to estimate the number of low-weight codewords in other relevant codes of rate  $1/2$ , including polar codes (with and without CRC precoding), the self-dual Reed-Muller code, and the PAC code with polar rate-profile. Our results are compiled in Table 1.



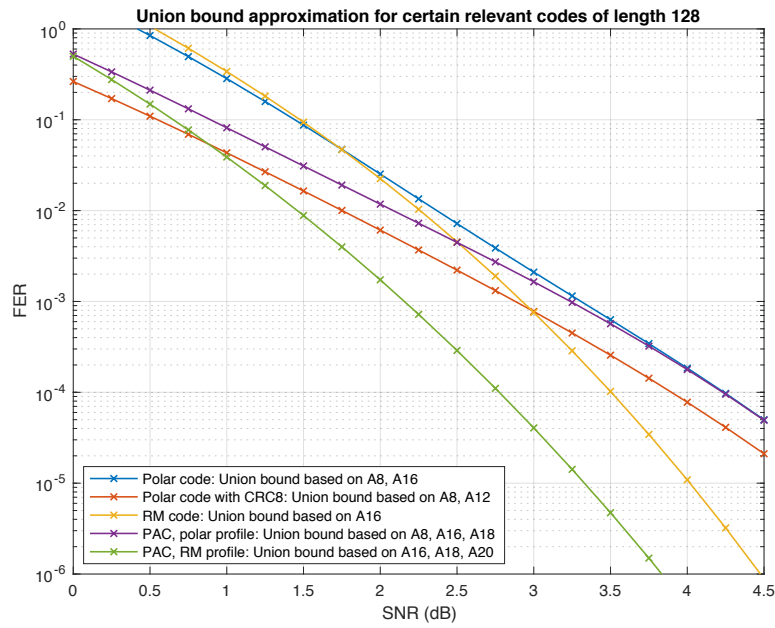
**Figure 8.** Low-weight codewords in the  $(128, 64)$  PAC code

	$A_8$	$A_{12}$	$A_{16}$	$A_{18}$	$A_{20}$	$A_{22}$
Polar code	48	0	68856	0	897024	0
Polar code, CRC8	20	173	$\geq 7069$	-	-	-
Reed-Muller	0	0	94488	0	0	0
PAC, polar profile	48	0	11032	6024	$> 10^5$	-
PAC, RM profile	0	0	3120	2696	95828	$> 10^5$

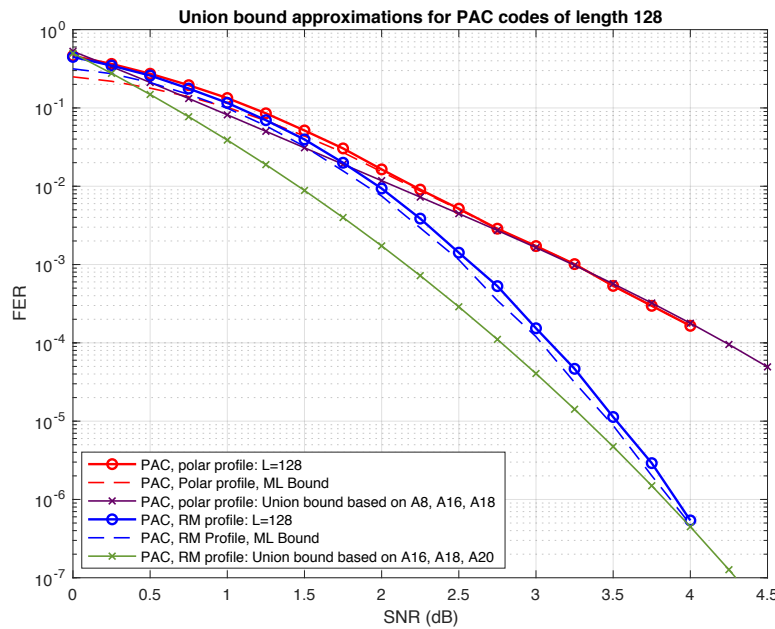
**Table 1.** Number of low-weight codewords in certain relevant codes

Again, the numbers in Table 1 should be regarded as lower bounds, which we conjecture to be exact (except for the Reed-Muller code whose weight distribution is known [39] and the polar code whose weight distribution can be computed using the methods of [55]). Assuming this conjecture, we expect the performance of the  $(128, 64)$  PAC code under ML decoding to be superior to all other polar and PAC codes in the table, since its minimum distance is twice as high. Interestingly, this code is also superior to the self-dual Reed-Muller code. The two codes have the same minimum distance, but the PAC code has significantly less codewords at this distance (by a factor of about 30).

These observations are corroborated in Figure 9, where we plot the truncated union bound based on the partial weight distributions compiled in Table 1 (with all other terms set to zero).



**Figure 9.** Truncated union bound for certain codes of length 128



**Figure 10.** Truncated union bound vs. performance for two PAC codes

It is well known that the performance of a linear code under ML decoding is governed by its weight distribution, and can be well approximated by the union bound or variants thereof [38], especially at high SNRs. The “truncated union bound” is by far the simplest option, obtained by

simply ignoring those terms in the union bound for which the weight distribution is unknown. Consequently, it is neither an upper bound nor a lower bound. Nevertheless, we have found that in the high SNR regime, it provides a reasonable first-order approximation of performance under ML decoding for the codes at hand. For example, Figure 10 shows the truncated union bound for the two PAC codes in Table 1 along with upper and lower bounds on their performance (under ML decoding) obtained in simulations.

Our results in this section also provide potential guidance for the difficult problem of PAC code design. Since both sequential decoding and list decoding achieve near-ML performance, one important goal of rate-profiling should be to optimize the weight distribution at low weights. The same criterion applies for the choice of the convolutional precoder as well.

As we can see from Table 1, the (128,64) PAC code with RM rate-profile succeeds at maintaining the minimum distance  $d = 16$  of the self-dual Reed-Muller code, while “shifting” most of the codewords of weight 16 to higher weights. This is another reason for the remarkable performance of this code. The fact that the minimum distance of this PAC code is  $d = 16$  also follows from [21, Theorem 1]. In fact, the work of Li, Zhang, and Gu [21] shows that precoding with *any nonsingular upper-triangular matrix*, not necessarily a Toeplitz matrix as in (2), cannot decrease the minimum distance. Moreover, there always exist such upper-triangular precoding matrices that *strictly reduce* the number of minimum-weight codewords [21, Theorem 2]. Apparently, the Toeplitz matrix generated by  $c = (1, 0, 1, 1, 0, 1, 1)$  is a particularly “nice” choice, reducing  $A_{16}$  from 94488 to only 3120. As we shall see in the next section, there are many such “nice” matrices, and it is possible to do even better.

## VI. PAC CODES WITH RANDOM TIME-VARYING CONVOLUTIONAL PRECODING

With reference to Section II, the two main considerations when designing the rate-1 convolutional precoder are: the constraint length  $\nu$  and the choice of the generator  $c = (c_0, c_1, \dots, c_\nu)$ . Arikan [3] refers to such generator  $c$  as the *impulse response* of the convolutional precoder. He furthermore writes in [3] that:

*As long as the constraint length of the convolution is sufficiently large, choosing  $c$  at random may be an acceptable design practice.*

The main question we wish to address herein is this: How large is “sufficiently large” in this context? It appears that if the impulse response  $c$  is fixed, then constraint length on the order

of  $\nu = 6$  is required. However, if we *allow the impulse response to vary with time*, then essentially the same performance can be achieved with constraint length as low as  $\nu = 2$  (which is the minimum possible, since  $c_0 = c_\nu = 1$  by assumption). This observation is of importance if trellis methods (such as list-Viterbi decoding, as suggested in [3], [35]) are used to decode PAC codes. Indeed, reducing the constraint length from  $\nu = 6$  to  $\nu = 2$  reduces the number of states in the resulting trellis from 64 to 4, respectively.

We also observe that under random time-varying convolutional precoding, the performance of PAC codes improves with constraint length (cf. Table 2) but only slightly.

#### A. Random Time-Varying Convolutional Precoding

In time-varying convolutional precoding, the impulse response  $\mathbf{c}$  is a function of time. Specifically, we fix the constraint length  $\nu$  but use  $n$  potentially different impulse response vectors:

$$\mathbf{c}^i = (c_0^i, c_1^i, \dots, c_\nu^i) \quad \text{for } i = 0, 1, \dots, n-1$$

where  $c_0^i = c_\nu^i = 1$  for all  $i$ . Thus each bit  $u_i$  of the input  $\mathbf{u} = (u_0, u_1, \dots, u_{n-1})$  to the polar encoder is computed via a different convolution operation:

$$u_i = \sum_{j=0}^{\nu} c_j^{i-j} v_{i-j} \quad \text{for } i = 0, 1, \dots, n-1 \quad (8)$$

where  $\mathbf{v}$  is the data-carrier vector resulting from the rate-profiling step, as in Section II. As before, the convolution operations in (8) can be recast a vector-matrix multiplication  $\mathbf{u} = \mathbf{v}\mathbf{T}$ , where  $\mathbf{T}$  is the following upper-triangular matrix:

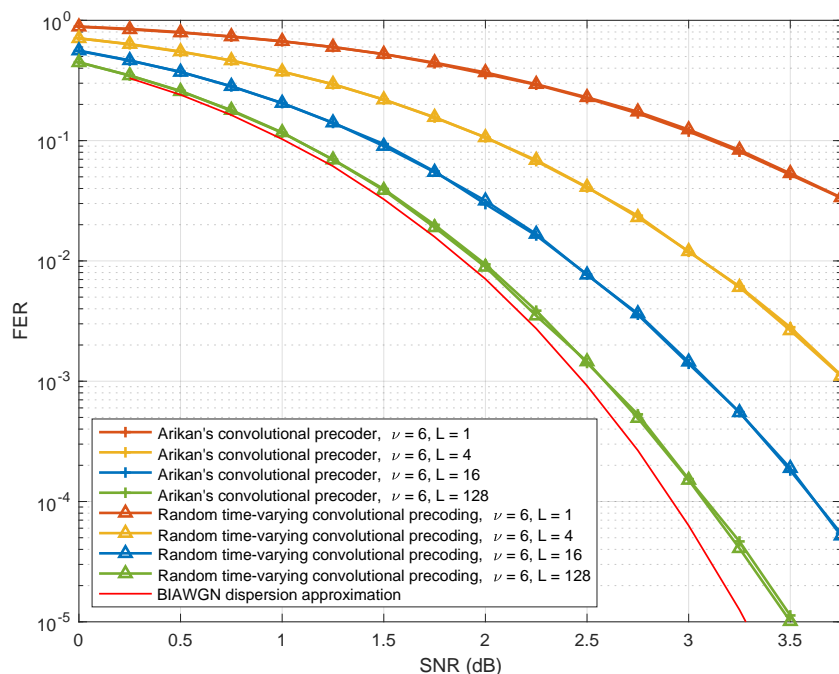
$$\mathbf{T} = \begin{bmatrix} c_0^0 & c_1^0 & c_2^0 & \dots & c_\nu^0 & 0 & \dots & 0 \\ 0 & c_0^1 & c_1^1 & c_2^1 & \dots & c_\nu^1 & & \vdots \\ 0 & 0 & c_0^2 & c_1^2 & \dots & \dots & c_\nu^2 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & & & \ddots & 0 & c_0^{n-3} & c_1^{n-3} & c_2^{n-3} \\ \vdots & & & & 0 & 0 & c_0^{n-2} & c_1^{n-2} \\ 0 & \dots & \dots & \dots & \dots & 0 & 0 & c_0^{n-1} \end{bmatrix} \quad (9)$$

In (9), the  $2n - \nu$  bits shown in red, namely  $c_0^i$  and  $c_\nu^i$ , are set to 1, whereas the  $(2n - \nu)(\nu - 1)/2$  bits shown in blue are unconstrained. In what follows, we consider *random* time-varying convolutional precoding, where these unconstrained bits are i.i.d. Bernoulli( $1/2$ ) random variables. That is, each of these bits is set to 0 or 1 with probability  $1/2$ , independently of each other.

On the decoder side, we use a straightforward modification of the list-decoding algorithm introduced in Section III. With reference to the pseudocode in Section III, this modification consists of replacing  $c_j$  by  $c_j^{\phi^{-j}}$  at line 14 of Algorithm 1 and at lines 9, 15, 18, 22 of Algorithm 2, where  $c_0^{\phi}, c_1^{\phi^{-1}}, \dots, c_v^{\phi^{-v}}$  are as defined in (9). The complexity of such modified list-decoding algorithm is exactly the same as before; the only difference being that the decoder now needs to store the  $n$  impulse responses  $c^0, c^1, \dots, c^{n-1}$ . However, this storage requirement is still linear in  $n$ .

### B. Performance of PAC codes with Random Time-Varying Convolutional Precoding

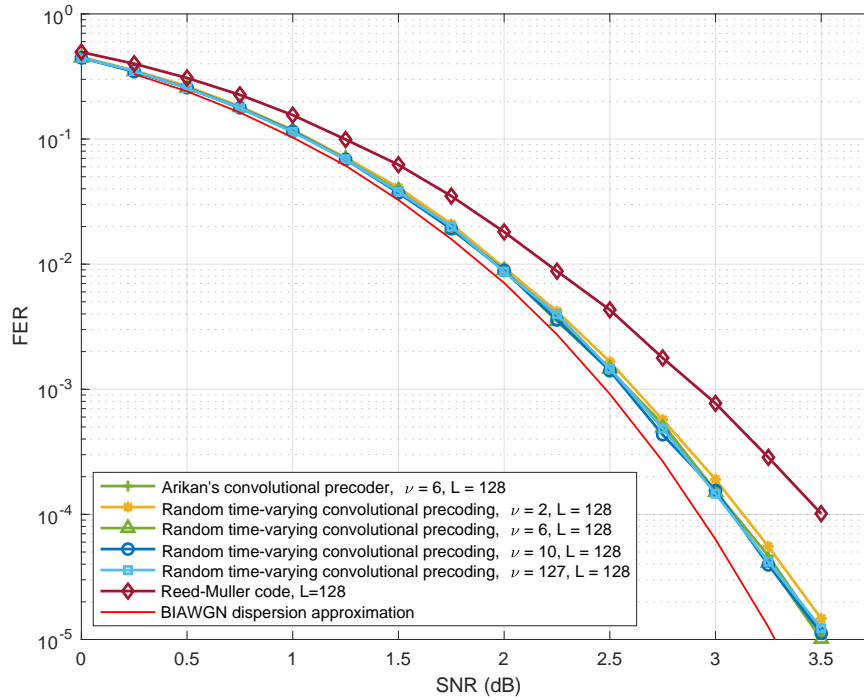
We now assess the performance of random time-varying convolutional precoding using our running example: the (128, 64) PAC code with RM rate profile. As the comparison benchmark, we employ the convolutional precoder with  $\nu = 6$  and  $\mathbf{c} = (1, 0, 1, 1, 0, 1, 1)$  used by Arikan in [3].



**Figure 11.** Performance of PAC codes with random time-varying precoding as a function of the list size

Figure 11 summarizes our simulation results for the case where the constraint length is fixed at  $\nu = 6$  while the list size ranges through  $L = 1, 4, 16, 128$ . We can see from this figure that the performance of PAC codes under random time-varying convolutional precoding coincides with the list-decoding performance of the benchmark for all the relevant list sizes.





**Figure 12.** Performance of PAC codes with random time-varying precoding as a function of the constraint length

	$A_8$	$A_{16}$	$A_{18}$	$A_{20}$	$A_{22}$
Random precoding with $\nu = 2$	0	6424	7780	142618	$> 10^5$
Arkan's PAC code with $\nu = 6$	0	3120	2696	95828	$> 10^5$
Random precoding with $\nu = 6$	0	2870	1526	88250	$> 10^5$
Random precoding with $\nu = 10$	0	2969	412	81026	$> 10^5$

**Table 2.** Number of low-weight codewords in PAC codes with random time-varying precoding

In Figure 12, we keep the list size constant at  $L = 128$ , but vary the constraint length  $\nu$ . Note that setting  $\nu = 0$  or  $\nu = 1$  leads to degenerate cases. For  $\nu = 0$ , the matrix in (9) reduces to the identity matrix and the resulting PAC code reduces to the  $(128, 64)$  Reed-Muller code; the performance of this Reed-Muller code is also shown in Figure 12, for comparison. For  $\nu = 1$ , the precoding matrix in (9) is *not* time-varying and *not* random, with each row being a shift of the vector  $\mathbf{c} = (1, 1)$ . Thus the smallest nontrivial constraint length is  $\nu = 2$ , which allows a single bit of randomness per row in (9). Somewhat surprisingly, this suffices to closely match the performance of Arkan's PAC code [3] with  $\nu = 6$ . As we increase the constraint length in (9) beyond  $\nu = 2$ , the performance further improves, but very slightly. Figure 12 shows that there is no significant gain even for  $\nu = 127$ , in which case the precoding matrix in (9) becomes

a random nonsingular upper-triangular matrix. In Table 2, we compile (the lower bounds on) the weight distribution for several typical realizations of the matrix in (9) which correspond to  $\nu = 2, 6, 10$ . These results corroborate the performance we observe in simulations.

## VII. CONCLUSIONS AND DISCUSSION

In this paper, we first observe that Arıkan’s PAC codes can be regarded as polar codes with dynamically frozen bits and then, using this observation, propose an efficient list decoding algorithm for PAC codes. We show that replacing sequential decoding of PAC codes by list decoding does not lead to degradation in performance, providing the list size is sufficiently large. We then carry out a qualitative complexity analysis of the two approaches, which suggests that list decoding may be advantageous in terms of worst-case complexity. We also study the performance of PAC codes (and other codes) under ML decoding by estimating the first few terms in their weight distribution. The results of this study provide constructive insights into the remarkable performance of PAC codes at short blocklengths. We furthermore introduce random time-varying convolutional precoding for PAC codes, and observe that this makes it possible to achieve the same remarkable performance with much smaller constraint length.

Based upon our results in this paper, we believe further complexity analysis of both sequential decoding and list decoding of PAC codes is warranted, including implementations in hardware. Some progress along these lines has been already reported in the recent paper [58], which uses the list-decoding algorithm introduced herein as a starting point. Indeed, we hope that our work stimulates further research in this direction.

Finally, we would like to point out two important (and interdependent) but difficult questions regarding PAC codes that remain open. What is the best choice of the rate profile? What is the best choice of the precoder? We hope our results will contribute to further study of these problems. In turn, effective resolution of these problems should make it possible to replicate the success of PAC codes at length  $n = 128$  for higher blocklengths.

## ACKNOWLEDGEMENT

We are grateful to Erdal Arıkan for tremendous help with this work. Specifically, we are indebted to him for sharing the data and the source code of his sequential decoding program.

## REFERENCES

- [1] E. Abbe and M. Ye, “Reed-Muller codes polarize,” preprint available as [arXiv:1901.11533](#), January 2019.
- [2] E. Arıkan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Transactions on Information Theory*, vol. 55, pp. 3051–3073, 2009.
- [3] E. Arıkan, “From sequential decoding to channel polarization and back again,” preprint available as [arXiv:1908.09594](#), September 2019.
- [4] E. Arıkan, “Systematic encoding and shortening of PAC Codes,” *Entropy*, vol. 22, no. 11, pp. 1301–1304, November 2020.
- [5] A. Balatsoukas-Stimming, *private communication*, August 2019.
- [6] M.C. Coşkun, G. Durisi, T. Jerkovits, G.L. Liva, W. Ryan, B. Stein, and F. Steiner, “Efficient error-correcting codes in the short blocklength regime,” *Physical Communication*, vol. 34, pp. 66–79, June 2019.
- [7] M.C. Coşkun, J. Neu, and H.D. Pfister, “Successive cancellation inactivation decoding for modified Reed-Muller and eBCH codes,” *Proc. IEEE International Symposium on Information Theory*, pp. 437–442, Los Angeles, CA, USA, June 2020.
- [8] T. Erseghe, “Coding in the finite-blocklength regime: Bounds based on Laplace integrals and their asymptotic approximations,” *IEEE Transactions on Information Theory*, vol. 62, pp. 6854–6883, December 2016.
- [9] R. Fano, “A heuristic discussion of probabilistic decoding,” *IEEE Transactions on Information Theory*, vol. 9, pp. 64–74, April 1963.
- [10] A. Fazeli, H. Hassani, M. Mondelli, and A. Vardy, “Binary linear codes with optimal scaling: Polar codes with large kernels,” *IEEE Transactions on Information Theory*, accepted for publication, October 2020.
- [11] A. Fazeli, K. Tian, and A. Vardy, “Viterbi-aided successive-cancellation decoding of polar codes,” *Proc. IEEE Global Communications Conference*, pp. 1–6, Singapore, December 2017.
- [12] A. Fazeli and A. Vardy, “On the scaling exponent of binary polarization kernels,” *Proc. Allerton Conf. Commun., Control, and Computing*, pp. 797–804, Monticello, IL, USA, October 2014.
- [13] A. Fazeli, A. Vardy and H. Yao, “Convolutional decoding of polar codes,” *Proc. IEEE International Symposium on Information Theory*, pp. 1397–1401, Paris, France, July 2019.
- [14] A.J. Ferris and D. Poulin, “Branching MERA codes: A natural extension of polar codes,” preprint available as [arXiv:1312.4575](#), December 2013.
- [15] A.J. Ferris, C. Hirche, and D. Poulin, “Convolutional polar codes,” preprint available as [arXiv:1704.00715](#), April 2017.
- [16] R.G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.
- [17] D. Goldin and D. Burshtein, “Performance bounds of concatenated polar coding schemes,” *IEEE Transactions on Information Theory*, vol. 65, pp. 7131–7148, November 2019.
- [18] S.B. Korada, E. Şaşıoğlu, and R.L. Urbanke, “Polar codes: Characterization of exponent, bounds, and constructions,” *IEEE Transactions on Information Theory*, vol. 56, pp. 6253–6264, December 2010.
- [19] 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, December 2012.
- [20] B. Li, H. Shen, and D. Tse, “RM-polar codes,” preprint available as [arXiv:1407.5483](#), July 2014.
- [21] B. Li, H. Zhang, J. Gu, “On pre-transformed polar codes,” preprint available as [arXiv:1912.06359](#), December 2019.
- [22] V. Miloslavskaya and P. Trifonov, “Sequential decoding of polar codes,” *IEEE Communications Letters*, vol. 16, no. 7, pp. 1127–1130, July 2014.
- [23] V. Miloslavskaya and B. Vucetic, “Design of short polar codes for SCL decoding,” *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6657–6668, November 2021.
- [24] S.K. Mishra and K.C. Kim, “Selectively precoded polar codes,” preprint available as [arXiv:2011.04930](#), Nov. 2020.

- [25] 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, December 2020.
- [26] M. Moradi, A. Mozammel, K. Qin, and E. Arkan, "Performance and complexity of sequential decoding of PAC codes," preprint available as [arXiv:2012.04990v1](#), December 2020.
- [27] A. Mozammel, "Hardware implementation of Fano decoder for PAC codes," preprint available as [arXiv:2011.09819](#), November 2020.
- [28] R. Morozov, "Convolutional polar kernels," *IEEE Transactions on Communications*, vol. 68, no. 12, pp. 7352–7361, December 2020.
- [29] R. Morozov and P. Trifonov, "Successive and two-stage systematic encoding of polar subcodes," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 877–880, June 2019.
- [30] R. Morozov and P. Trifonov, "On distance properties of convolutional polar codes," *IEEE Transactions on Communications*, vol. 67, no. 7, pp. 4585–4592, July 2019.
- [31] E. Moskovskaya and P. Trifonov, "Design of BCH polarization kernels with reduced processing complexity," *IEEE Communications Letters*, vol. 24, no. 7, pp. 1383–1386, July 2020.
- [32] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, October 2012.
- [33] Y. Polyanskiy, H.V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, pp. 2307–2359, May 2010.
- [34] M. Rowshan, A. Burg, and E. Viterbo, "Polarization-adjusted convolutional (PAC) codes: Fano decoding vs list decoding," preprint available as [arXiv:2002.06805v1](#), February 2020.
- [35] M. Rowshan and E. Viterbo, "List Viterbi decoding of PAC codes," preprint available as [arXiv:2007.05353](#), July 2020.
- [36] G. Sarkis, P. Giard, A. Vardy, C. Thibault, and W.J. Gross, "Increasing the speed of polar list decoders," *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, Belfast, UK, October 2014.
- [37] G. Sarkis, P. Giard, A. Vardy, C. Thibault, and W.J. Gross, "Fast list decoders for polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 318–328, February 2016.
- [38] I. Sason and S. Shamai, "Performance analysis of linear codes under maximum-likelihood decoding: A tutorial," *Foundations and Trends in Communications and Information Theory*, vol. 3, nos. 1–2, pp. 1–225, NOW Publishers, July 2006.
- [39] M. Sugino, Y. Ienaga, Y. Tokura, and T. Kasami, "Weight distribution of (128, 64) Reed-Muller code," *IEEE Transactions on Information Theory*, vol. 17, pp. 627–628, September 1971.
- [40] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.
- [41] T. Tonnellier and W.J. Gross, "On systematic polarization-adjusted convolutional (PAC) codes," preprint available as [arXiv:2011.03177](#), November 2020.
- [42] P. Trifonov, "Star polar subcodes," *Proc. IEEE Wireless Communications and Networking Conference*, pp. 1–6, San Francisco, CA, USA, March 2017.
- [43] P. Trifonov, "A score function for sequential decoding of polar codes," *Proc. IEEE International Symposium on Information Theory*, pp. 1470–1474, Vail, CO, USA, June 2018.
- [44] P. Trifonov, "On construction of polar subcodes with large kernels," *Proc. IEEE International Symposium on Information Theory*, pp. 1932–1936, Paris, France, July 2019.
- [45] P. Trifonov, "Trellis-based decoding techniques for polar codes with large kernels," *Proc. IEEE Information Theory Workshop*, pp. 1–5, Visby, Sweden, August 2019.

- [46] P. Trifonov, "Randomized polar subcodes with optimized error coefficient," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6714–6722, November 2020.
- [47] P. Trifonov and V. Miloslavskaya, "Polar codes with dynamic frozen symbols and their decoding by directed search," *Proc. IEEE Information Theory Workshop*, pp. 1–5, Sevilla, Spain, September 2013.
- [48] P. Trifonov and V. Miloslavskaya, "Polar subcodes," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 254–266, February 2016.
- [49] P. Trifonov and G. Trofimiuk, "A randomized construction of polar subcodes," *Proc. IEEE International Symposium on Information Theory*, pp. 1863–1867, Aachen, Germany, July 2017.
- [50] G. Trofimiuk and P. Trifonov, "Efficient decoding of polar codes with some  $16 \times 16$  kernels," *Proc. IEEE Information Theory Workshop*, pp. 11–15, Guangzhou, China, November 2018.
- [51] G. Trofimiuk and P. Trifonov, "Reduced complexity window processing of binary polarization kernels," *Proc. IEEE International Symposium on Information Theory*, pp. 1412–1416, Paris, France, July 2019.
- [52] L. Wang, M. Jiang, C. Zhao, and Z. Li, "Genetic optimization of short block-length PAC codes for high capacity PHz communications," SPIE vol. 11617, *Proc. International Conference on Optoelectronic and Microelectronic Technology and Application*, Nanjing, China, December 2020.
- [53] H. Yao, A. Fazeli, and A. Vardy, "Explicit polar codes with small scaling exponent," *Proc. IEEE International Symposium on Information Theory*, pp. 1757–1761, Paris, France, July 2019.
- [54] H. Yao, A. Fazeli, and A. Vardy, "List decoding of Arkan's PAC codes," *Proc. IEEE International Symposium on Information Theory*, pp. 443–448, Los Angeles, CA, USA, June 2020.
- [55] H. Yao, A. Fazeli, and A. Vardy, "A deterministic algorithm for computing the weight distribution of polar codes," preprint available as [arXiv:2102.07362](https://arxiv.org/abs/2102.07362), February 2021.
- [56] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed-Muller codes" preprint available as [arXiv:1902.01470](https://arxiv.org/abs/1902.01470), February 2019.
- [57] P. Yuan, T. Prinz, G. Böcherer, O. Iscan, R. Boehnke, and W. Xu, "Polar code construction for list decoding," *12-th International ITG Conference on Systems, Communications and Coding*, pp. 1–6, Rostock, Germany, February 2019.
- [58] H. Zhu, Z. Cao, Y. Zhao, D. Li, Y. Yang, and Y. Wang, "Fast list decoders for polarization-adjusted convolutional (PAC) codes," preprint available as [arXiv:20122.09425](https://arxiv.org/abs/20122.09425), December 2020.
- [59] 3GPP TSG RAN WG1 Meeting #87, Final report, Reno, NV, November 2016.
- [60] 3GPP Technical Specification Group Radio Access Network, "Multiplexing and channel coding" Release 16, 3GPP TS 38.212 V16.3.0, September 2020.