entropy

Article

List Decoding of Arıkan's PAC Codes †

Hanwen Yao, Arman Fazeli and Alexander Vardy *

Department of Electrical and Computer Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA; hwyao@ucsd.edu (H.Y.); afazelic@ucsd.edu (A.F.)
* Correspondence: avardy@ucsd.edu

† This work was presented in part at the *IEEE International Symposium on Information Theory*, Los Angeles, CA, USA, 21–26 June 2020.

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. However, its performance at short blocklengths under standard successive cancellation decoding is far from optimal. A well-known way to improve the performance of polar codes at short blocklengths is CRC precoding followed by successive-cancellation list decoding. This approach, along with various refinements thereof, has largely remained the state of the art in polar coding since it was introduced in 2011. Recently, Arıkan 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. One of our primary 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, $L \geqslant 128$). 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 $\nu = 2$.

Keywords: coding theory; polar codes; convolutional codes; list decoding; sequential decoding

check for updates

Citation: Yao, H.; Fazeli, A.; Vardy, A. List Decoding of Arıkan's PAC Codes. *Entropy* **2021**, *23*, 841. https://doi.org/10.3390/e23070841

Academic Editor: Mahyar Shirvanimoghaddam

Received: 10 May 2021
Accepted: 16 June 2021
Published: 30 June 2021

1. Introduction

Polar coding, pioneered by Arıkan [1], 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.

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 Arıkan in [2]. Codes of length 128 and rate $1/2$ serve as the running example throughout Arıkan's recent paper [2], and we will also adopt this strategy herein. We make no attempt to optimize these codes; rather, our goal is to follow Arıkan [2] 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 [3]).

Clearly, at length 128, there is a tremendous gap between polar codes under SC decoding and the best achievable performance.

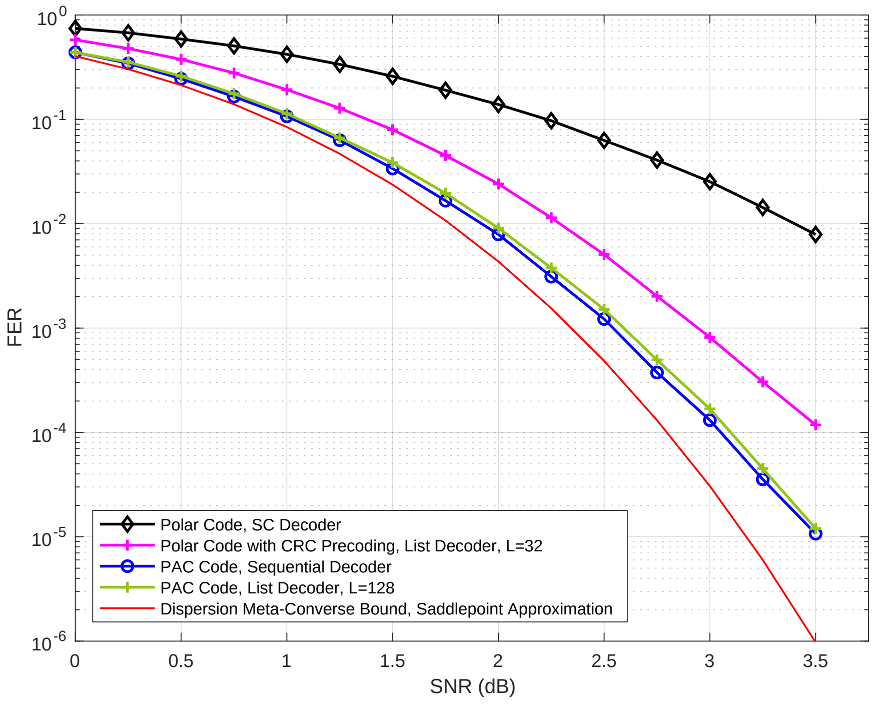

Figure 1. Performance of PAC codes versus polar codes.

As shown in [4] 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 [2], 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 [5–7] and other papers), has largely remained the state of the art in polar coding since it was first introduced in [4]. It is currently used as the coding scheme for control and physical broadcast channels in the enhanced mobile broadband (eMBB) mode and the ultra-reliable low latency communications (URLLC) mode of the fifth generation (5G) wireless communications standard [8].

In the Shannon Lecture at the ISIT in 2019, Erdal Arıkan presented a significant breakthrough in polar coding, which significantly boosts the performance of polar codes at short lengths. Specifically, Arıkan [2] 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 [3,9]. The performance of PAC codes of length 128 and rate $1/2$ is also shown (in blue and green) in Figure 1.

1.1. Brief Overview of PAC Codes

Arıkan's PAC codes 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 [2].

Referring to Figure 2, let us consider an (n, k) PAC code. On the encoding side, Arıkan uses a rate-1 convolutional precoder concatenated with a standard polar encoder. Only k out of the n bits of the input $\boldsymbol{v}$ to the convolutional precoder carry the information (or data) vector $\boldsymbol{d}$. The remaining $n - k$ bits of $\boldsymbol{v}$ are set to 0. Just like for conventional polar codes, the overall performance of the resulting PAC code crucially depends upon which positions in $\boldsymbol{v}$ carry information and which are frozen to 0. This choice of frozen positions in $\boldsymbol{v}$, Arıkan 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 5.

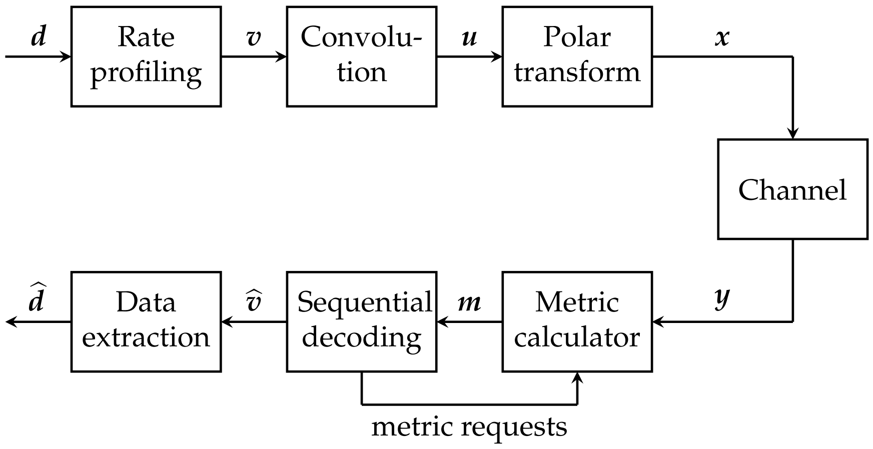

Figure 2. PAC coding scheme.

The main operation on the decoder side is sequential decoding. Specifically, Arıkan employs Fano decoding (as described in [10] and in Section 6.9 of [11]) of the convolutional code to estimate its input $\boldsymbol{v}$. The path metrics used by this sequential decoder are obtained via repeated calls to the successive-cancellation decoder for the underlying polar code.

1.2. 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 4. 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 [5] to estimate the number of low-weight codewords in PAC codes, under polar and RM rate profiles (introduced by Arıkan [2]). 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 3 and Figures 9 and 10. We also introduce and study random time-varying convolutional precoding for PAC codes. We find that, as compared with the convolutional precoding introduced in [2], time-varying convolutional precoding is much less sensitive to the constraint length. Arıkan uses in [2] a convolutional code generated by $\boldsymbol{c} = (1, 0, 1, 1, 0, 1, 1)$, 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 $\nu = 2$.

1.3. 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 [4], adaptive list decoding [5], sequential decoding [6,12], and stack decoding [7], 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 [4]. However, many other modifications of the basic polar-coding paradigm have been extensively studied, including large polarization kernels [13–21], polar subcodes [17,22–27], "convolutional" polar codes [28–31], and polarized Reed–Muller coding [32–35] among others.

As shown later in this paper, in Arıkan'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 [25], and later studied in [17,22,23,25–27,36–38] and other papers. However, the dynamic freezing patterns in these papers are very different from [2]. Prior to Arıkan's work [2], convolutional precoding of polar codes was proposed in [39] and later studied in [40].

Although quite recent, Arıkan's PAC codes have already attracted considerable interest; see for example [41–47]. While these papers investigate various aspects of PAC codes, none of them considers list decoding thereof. Finally, we note the work of [48,49], which investigates both Fano decoding and list decoding of PAC codes. This work is apparently independent from and contemporaneous with our results herein. The paper of Rowshan, Burg, and Viterbo [48] was posted on `arxiv.org` in February 2020, while our work [50] was submitted for review in January 2020. Our results became available on `arxiv.org` in May 2020. The Rowshan-Viterbo paper [49] was posted on `arxiv.org` in July 2020, after our results were presented in [50].

1.4. Paper Outline

The rest of this paper is organized as follows. We begin with an overview on Arıkan's PAC codes in Section 2, including both their encoding process and sequential decoding. In Section 3, we present our list-decoding algorithm. In Section 4, we compare it with sequential decoding, in terms of both performance and complexity. In Section 5, 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 6, 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 7.

2. Overview of Arıkan's PAC Codes

For details on conventional polar codes under standard SC decoding, we refer the reader to Arıkan's seminal paper [1]. Like polar codes, the block length n of a PAC code is also a power of 2. That is, $n = 2^m$ with $m \geqslant 1$. 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 $\boldsymbol{d}$ are embedded into a data-carrier vector $\boldsymbol{v}$ of length n, at k positions specified by an index set $\mathcal{A} \subseteq \{0, 1, \ldots, n-1\}$ with $|\mathcal{A}| = k$. The remaining $n - k$ positions in $\boldsymbol{v}$ are frozen to zero. Arıkan [2] 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. Arıkan has proposed in [2] two alternative approaches for selecting this set $\mathcal{A}$. The first approach, called ***polar rate-profiling***, proceeds as follows. Let $W_0, W_1, \ldots, W_{n-1}$ 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 $I(W_0), I(W_1), \ldots, I(W_{n-1})$. The second approach proposed in [2] is called ***RM rate-profiling***. Let $\mathrm{wt}(i)$ 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 $\{\mathrm{wt}(i) : i \in \mathcal{A}\}$ consists of the k largest values among $\mathrm{wt}(0), \mathrm{wt}(1), \ldots, \mathrm{wt}(n-1)$. 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 $\boldsymbol{v}$ resulting from the rate-profiling step is encoded using a rate-1 convolutional code generated by $\boldsymbol{c} = (c_0, c_1, \ldots, c_\nu)$, with $c_0 = c_\nu = 1$ (the latter can be assumed without loss of generality). This produces another vector $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ 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 $\boldsymbol{u}$ is a linear combination of $(\nu + 1)$ bits of $\boldsymbol{v}$ computed via the convolution operation:

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

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

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

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

$$\mathbf{P}_m = \mathbf{B}_n \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes m} \tag{3}$$

where $\mathbf{B}_n$ is the $n \times n$ bit-reversal permutation matrix (as defined in Section VII of [1]), and $\mathbf{P}_m$ is known as the polar transform matrix. Alternatively, the polar transform can be defined as in (3) but without the bit-reversal matrix $\mathbf{B}_n$; this has no effect on the performance of the resulting codes.

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 $\boldsymbol{c} = (1,0,1,1,0,1,1)$. This produces the (128,64) PAC code, which is the code studied by Arıkan in [2]. This specific PAC code will serve as our primary running example throughout the paper.

On the decoding side, Arıkan [2] employs sequential decoding of the underlying convolutional code to decode the data-carrier vector $\boldsymbol{v}$. 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. Arıkan [2] uses the Fano sequential decoder, described in [10,11]. 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.

3. 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 (see Algorithms 1 and 2). In the next section, we will consider the performance and complexity of the resulting decoder, as compared to the sequential decoder of [2].

3.1. 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 [4]. The complexity of this algorithm is $O(Ln\log n)$, where L is the list size. However, the algorithm of [4] 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 [25], and later studied by the same authors in [26,27]. 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 $\mathcal{F} \subset \{0, 1, \ldots, n-1\}$ 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 [25], 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

$$u_i = f_i(u_0, u_1, \ldots, u_{i-1}) \tag{4}$$

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

$$\widehat{u}_i = f_i(\widehat{u}_0, \widehat{u}_1, \ldots, \widehat{u}_{i-1}) \tag{5}$$

where $\widehat{u}_0, \widehat{u}_1, \ldots, \widehat{u}_{i-1}$ are its earlier decisions. The encoding/decoding process in (4) and (5) is known as dynamic freezing.

In order to explain how Arıkan's PAC codes [2] fit into the dynamic freezing framework, let us first introduce some notation. With reference to Section 2, for $i = 0, 1, \ldots, n-1$, let $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ denote the vectors $(u_0, u_1, \ldots, u_i)$ and $(v_0, v_1, \ldots, v_i)$, respectively. Further, let $\mathbf{T}_{i,j}$ denote the submatrix of the Toepliz 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 $\boldsymbol{u}_i = \boldsymbol{v}_i\mathbf{T}_{i,i}$ 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 $\boldsymbol{v}_i = \boldsymbol{u}_i\mathbf{T}_{i,i}^{-1}$ for all i. Now suppose that $i \in \mathcal{A}^c$, so that v_i is frozen to zero in the rate-profiling step. Then we have

$$\boldsymbol{u}_i = \boldsymbol{v}_{i-1}\mathbf{T}_{i-1,i} = \left(\boldsymbol{u}_{i-1}\mathbf{T}_{i-1,i-1}^{-1}\right)\mathbf{T}_{i-1,i} \tag{6}$$

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

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

where $(0, \ldots, 0, c_\nu, c_{\nu-1}, \ldots, c_1)^t$ represents the last column of the matrix $\mathbf{T}_{i-1,i}$. Clearly, the above is a special case of dynamic freezing in (4).

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 $v_{i-1}, v_{i-2}, \ldots, v_{i-\nu}$ 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 (6). Instead, we can successively compute the vector $\widehat{\boldsymbol{v}} = (\widehat{v}_0, \widehat{v}_1, \ldots, \widehat{v}_{n-1})$ as follows. If $i \in \mathcal{A}^c$, set $\widehat{v}_i = 0$. Otherwise, set

$$\widehat{v}_i \;=\; \widehat{u}_i - \sum_{j=1}^{\nu} c_j \widehat{v}_{i-j} \tag{7}$$

where the value of $\widehat{u}_i$ is provided by the polar decoder. Given $\widehat{v}_i, \widehat{v}_{i-1}, \ldots, \widehat{v}_{i-\nu}$, the values of the dynamically frozen bits $\widehat{u}_i$ for $i \in \mathcal{A}^c$ can be computed using (1). This computation, along with the one in (7), takes time linear in ν. All that is required is additional memory to store the vector $\widehat{\boldsymbol{v}} = (\widehat{v}_0, \widehat{v}_1, \ldots, \widehat{v}_{n-1})$.

Algorithm 1: List Decoder for PAC Codes

```
Input: The received vector $\boldsymbol{y}$, the list size $L$, the generator $\boldsymbol{c} = (c_0, c_1, \ldots, c_\nu)$ for
       the convolutional precoder as global
Output: Decoded codeword $\widehat{\boldsymbol{x}}$
   // Initialization
1  ··· lines 2–5 of Algorithm 12 in [4]
2  shiftRegisters ← new 2-D array of size $L \times (\nu + 1)$
3  for $\ell = 0, 1, \ldots, L-1$ do
4      shiftRegister[$\ell$] = $(0, 0, \ldots, 0)$
   // Main Loop
5  for $\phi = 0, 1, \ldots, n-1$ do
6      recursivelyCalcP($m, \phi$)
7      if $u_\phi$ is frozen then
8          for $\ell = 0, 1, \ldots, L-1$ do
9              if activePath[$\ell$] = false then
10                 continue
11             left-shift shiftRegister[$\ell$] by one, with the rightmost position set to 0
12             $C_m$ ← getArrayPointerC($m, \ell$)
13             $(v_{\phi-\nu}, v_{\phi-\nu+1}, \ldots, v_\phi)$ ← shiftRegister[$\ell$]
               // Set the frozen bit
14             $C_m[0][\phi \bmod 2] \leftarrow \sum_{j=0}^{\nu} c_j v_{\phi-j}$
15     else
16         continuePaths_Unfzn($\phi$)
17     if $\phi \bmod 2 = 1$ then
18         recursivelyUpdateC($m, \phi$)
   // Get the best codeword in the list
19 ··· lines 17–24 of Algorithm 12 in [4]
20 return $\widehat{\boldsymbol{x}} = (C_0[\beta][0])_{\beta=0}^{n-1}$
```

3.2. *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.

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 $\widehat{u}_i = 0$. For PAC codes, since freezing is dynamic, different paths are potentially extended differently, depending upon the previous decisions along the path.

Algorithm 2: continuePaths_Unfzn (PAC version)

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

In general, our list decoder for PAC codes maintains the same data structure as the successive-cancellation list decoder in [4]. 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 ν bits of the vector $\widehat{v} = (\widehat{v}_0, \widehat{v}_1, \ldots, \widehat{v}_{n-1})$, computed as in (7), for the corresponding path.

Algorithms 1 and 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 [4], with the differences highlighted in blue.

4. 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 3. For sequential decoding, we employ exactly the same Fano decoder that was used by Arıkan in [2]. We are grateful to Erdal Arıkan for sharing the details of their decoding algorithm. We do not disclose these details here, instead referring the reader to [2,44,45].

We note that more efficient algorithms for sequential decoding of polar codes and their subcodes may be available; see in particular the work of Trifonov [12,27]. However, in this paper, we use the results of Arıkan [2] 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.

4.1. Performance Comparison

Figure 3 summarizes simulation results comparing the performance of the Fano decoder from [2] with our list decoding algorithm, as a function of the list size L. The underlying PAC code is the same as in Figure 1; it is the (128,64) PAC code obtained via RM rate-profiling (see Section 2). The underlying channel is the binary-input additive white Gaussian noise (BIAWGN) channel.

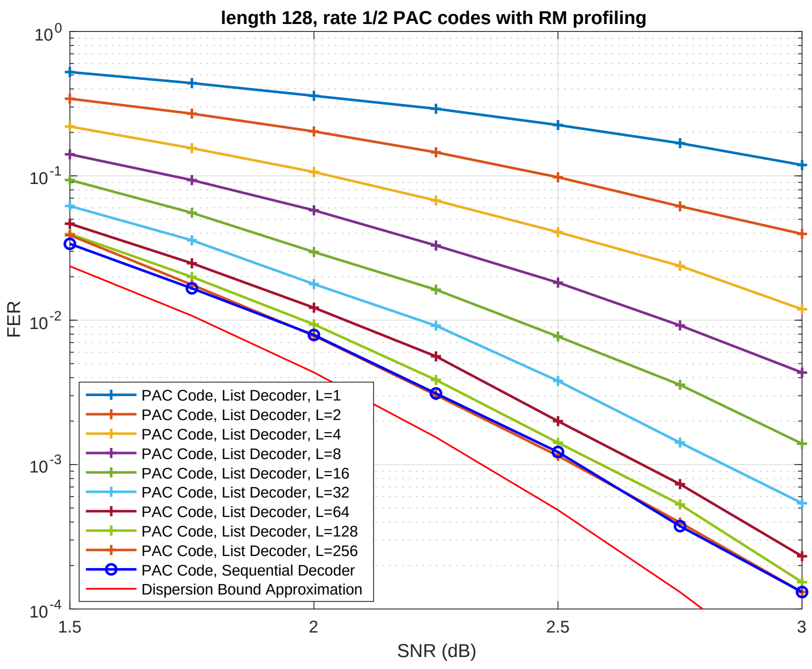

Figure 3. Performance of PAC codes under list decoding.

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 [2], 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. Table 1 below shows what fraction of such cases were due to decoding failures:

Table 1. Fraction of decoding failures as a function of SNR.

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 [2]. Overall, the foregoing table indicates that increasing this cap would not improve the performance significantly. In fact, we observe that decoding failures never dominate the overall FER of sequential decoding. Thus, it would be interesting to investigate how much this cap can be decreased without sacrificing the performance.

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. Table 2 below shows the fraction of selection errors for lists of various sizes:

Table 2. Fraction of selection errors as a function of SNR.

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 $L \geqslant 64$) 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 saddlepoint approximation [51] of the meta-converse dispersion bound of Polyanskiy, Poor, and Verdu [3]. Our curve coincides with those given in Figure 1 of [52] and Figure 6 of [53]. Note that a more accurate bound can be derived using the methods of Erseghe [9], 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 $L \geqslant 128$, is close to the best achievable performance.

4.2. 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 3.1) in conjunction with existing work on efficient hardware implementation of polar list decoders (see [54,55], for 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 [56].

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 $\boldsymbol{u} = (u_0, u_1, \ldots, u_{n-1})$ 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.

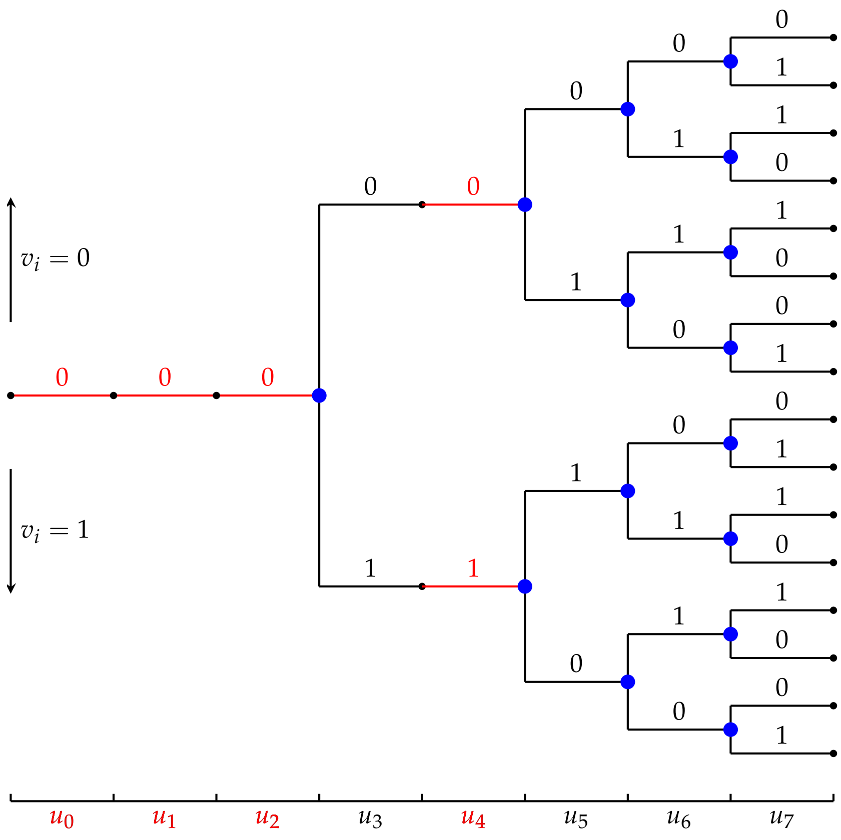

Figure 4. An example of the polar search tree, reproduced from [2].

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, counting 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 4.1). 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 $1 + 2 + 4 + \cdots + L = 2L - 1$. 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 $L(k + 2 - \log_2 L) - 1 = O(kL)$.

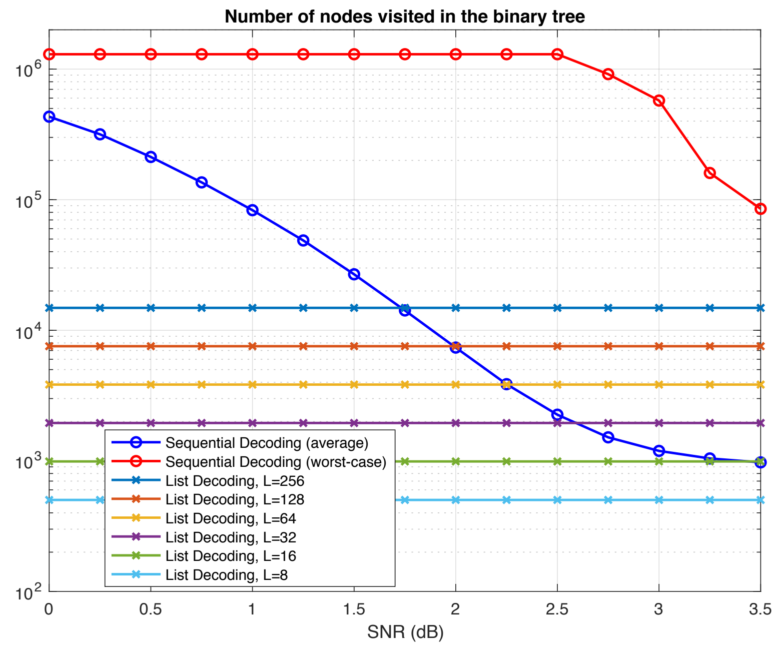

Figure 5. Sequential decoding vs. list decoding: Number of nodes visited in the polar search tree.

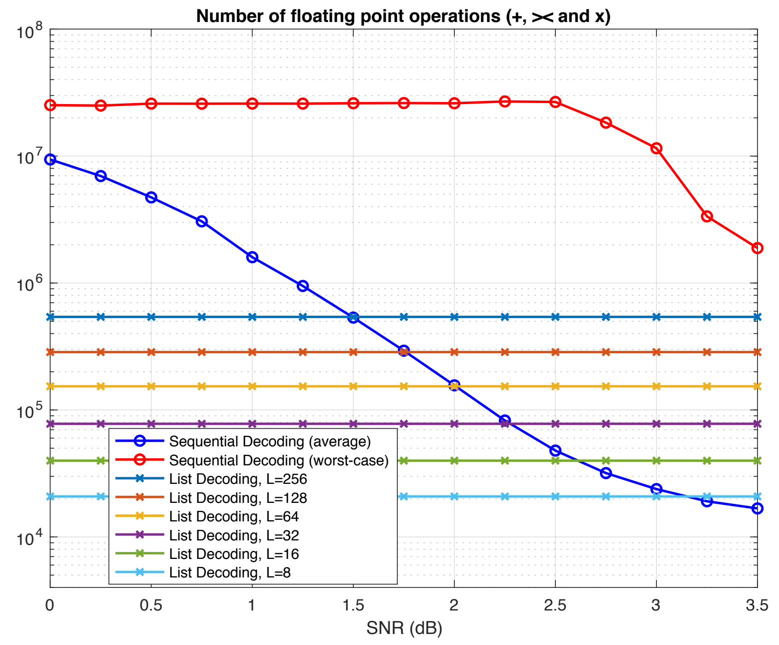

Figure 6. Sequential decoding vs. list decoding: Number of floating-point operations.

If L is not a power of 2, this counting argument readily generalizes, and the number of decision nodes visited is given by

$$L\big(k + 1 - \lceil \log_2 L \rceil\big) + 2^{\lceil \log_2 L \rceil} - 1 \;=\; O(kL) \tag{8}$$

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 (8), but the complexity is still independent 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.

5. 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.

5.1. 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 Arıkan'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 better path-metric (higher 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 [4] for polar codes, except that [4] used list decoding.

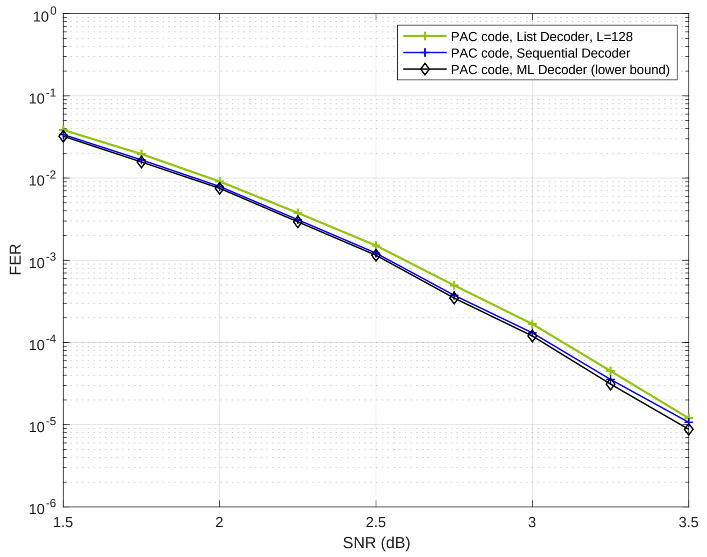

Figure 7. Performance of the PAC code under ML decoding.

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 Arıkan's PAC codes are so good at short blocklengths: they can be efficiently decoded with near-ML fidelity.

5.2. *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. The results of this study are summarized in Table 3 and in Figures 8–10.

First, consider the following experiment, devised in [5]. 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 16,18,20,22. Moreover, $A_{16} \geqslant 3120$, $A_{18} \geqslant 2696$, and $A_{20} \geqslant 95828$ (cf. Table 3). 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 3.

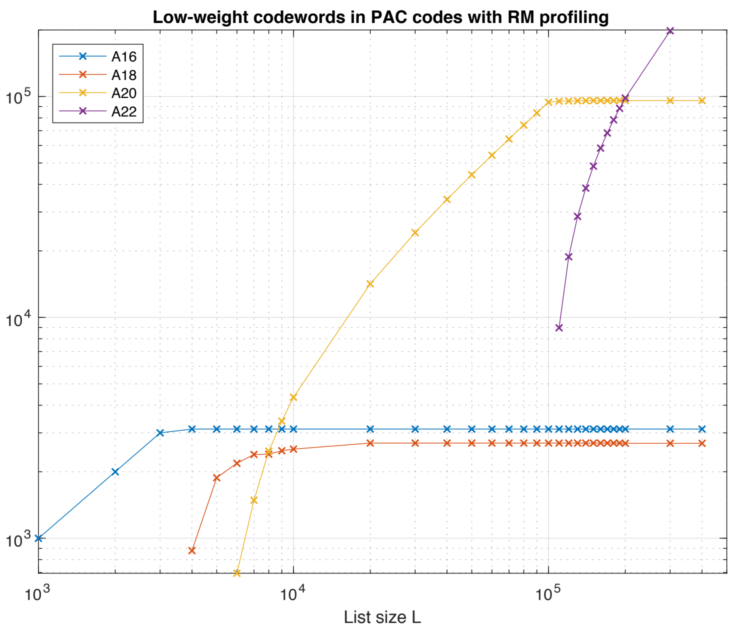

Figure 8. Low-weight codewords in the (128,64) PAC code.

Table 3. Number of low-weight codewords in certain relevant codes.

	A_8	A_{12}	A_{16}	A_{18}	A_{20}	A_{22}
Polar code	48	0	68,856	0	897,024	0
Polar code, CRC8	20	173	$\geqslant$7069	-	-	-
Reed–Muller	0	0	94,488	0	0	0
PAC, polar profile	48	0	11,032	6024	$>10^5$	-
PAC, RM profile	0	0	3120	2696	95,828	$>10^5$

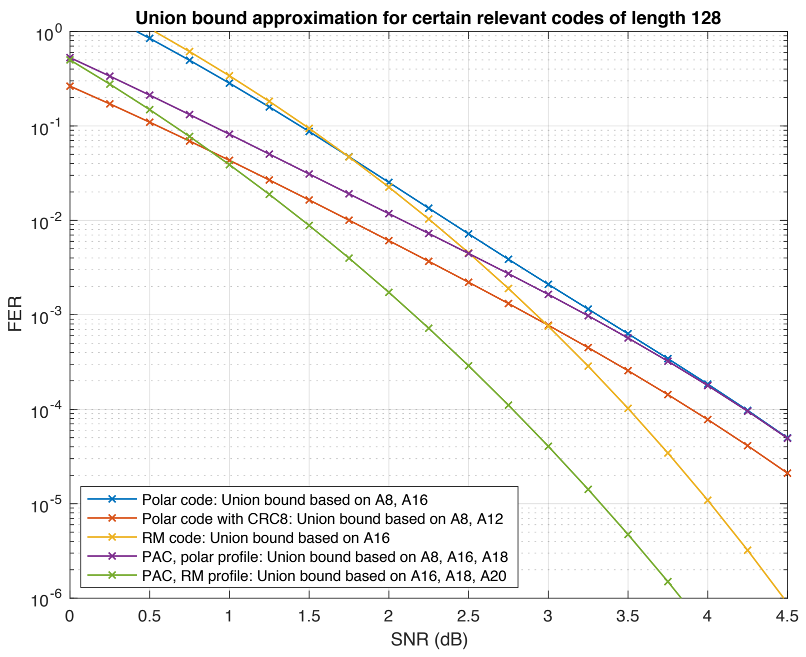

Figure 9. Truncated union bound for certain codes of length 128.

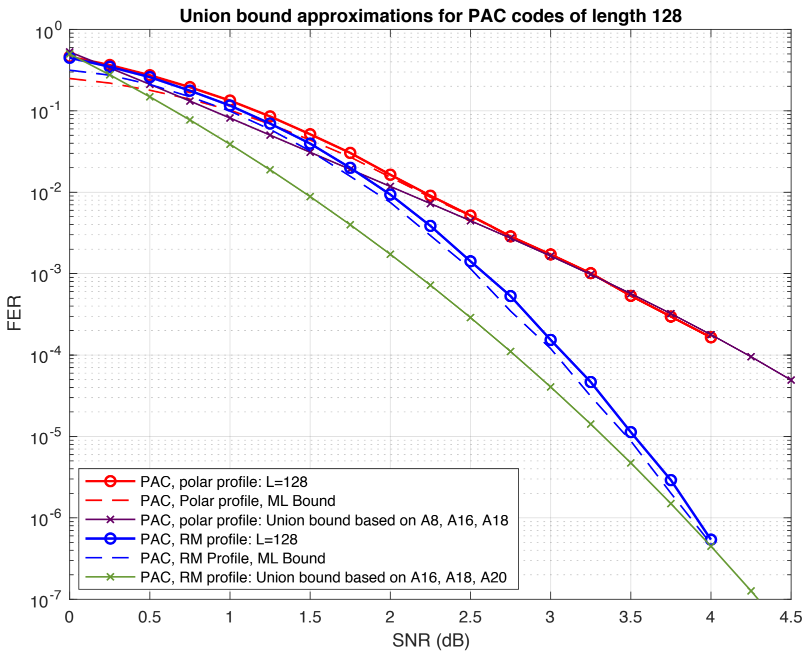

Figure 10. Truncated union bound vs. performance for two PAC codes.

Again, the numbers in Table 3 should be regarded as lower bounds, which we conjecture to be exact (except for the Reed–Muller code whose weight distribution is known [57] and the polar code whose weight distribution can be computed using the methods of [58]). Assuming this conjecture, we expect the performance under ML decoding of the (128,64) PAC code with RM rate-profile 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 Figures 9 and 10, where we plot the truncated union bound based on the partial weight distributions compiled in Table 3 (with all other terms set to zero). 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 [59], 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 3 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. A related problem is that of finding the best rate profile for a given list size, which does not necessarily approach ML decoding.

As we can see from Table 3, 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 Theorem 1 of [42]. In fact, the work of Li, Zhang, and Gu [42] shows that precoding with any nonsingular upper-triangular matrix, not necessarily a Toepliz matrix as in (2), cannot decrease the minimum distance. Moreover, there always exist such upper-triangular precoding matrices that strictly reduce the number of mimimum-weight codewords (see Theorem 2 of [42]). Apparently, the Toepliz matrix generated by $\boldsymbol{c} = (1, 0, 1, 1, 0, 1, 1)$ 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.

6. PAC Codes with Random Time-Varying Convolutional Precoding

With reference to Section 2, the two main considerations when designing the rate-1 convolutional precoder are: the constraint length ν and the choice of the generator $\boldsymbol{c} = (c_0, c_1, \ldots, c_\nu)$. Arıkan [2] refers to such generator $\boldsymbol{c}$ as the ***impulse response*** of the convolutional precoder. He furthermore writes in [2] that:

> *As long as the constraint length of the convolution is sufficiently large, choosing* $\boldsymbol{c}$ *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 $\boldsymbol{c}$ 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 [2,49]) 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 but only slightly.

6.1. Random Time-Varying Convolutional Precoding

In time-varying convolutional precoding, the impulse response $\boldsymbol{c}$ is a function of time. Specifically, we keep the constraint length ν fixed, but use n potentially different impulse response vectors $\boldsymbol{c}^i = (c_0^i, c_1^i, \ldots, c_\nu^i)$, where $c_0^i = c_\nu^i = 1$ for all i. Thus, each bit

u_i of the input $\boldsymbol{u} = (u_0, u_1, \cdots, u_{n-1})$ to the polar encoder is computed via a potentially different convolution operation:

$$u_i \;=\; \sum_{j=0}^{\nu} c_j^{i-j}\, v_{i-j} \qquad \text{for } i = 0, 1, \cdots, n-1 \tag{9}$$

where $\boldsymbol{v}$ is the data-carrier vector resulting from the rate-profiling step, as in Section 2. As before, the convolution operations in (9) can be recast a vector-matrix multiplication $\boldsymbol{u} = \boldsymbol{v}\mathbf{T}$, where $\mathbf{T}$ is the following upper-triangular matrix:

$$\mathbf{T} \;=\; \begin{bmatrix} c_0^0 & c_1^0 & c_2^0 & \cdots & c_\nu^0 & 0 & \cdots & 0 \\ 0 & c_0^1 & c_1^1 & c_2^1 & \cdots & c_\nu^1 & & \vdots \\ 0 & 0 & c_0^2 & c_1^2 & \ddots & \cdots & c_\nu^2 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & & & \ddots & 0 & c_0^{n-3} & c_1^{n-3} & c_2^{n-3} \\ \vdots & & & & 0 & 0 & c_0^{n-2} & c_1^{n-2} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 0 & c_0^{n-1} \end{bmatrix} \tag{10}$$

In (10), the $2n - \nu$ bits shown in red, namely c_0^i and c_ν^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$\left({}^1\!/_2\right)$ random variables. That is, each of these $(2n - \nu)(\nu - 1)/2$ 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 3. With reference to the pseudocode in Section 3, this modification consists of replacing c_j by $c_j^{\phi-j}$ at line 14 of Algorithm 1 as well as lines 9, 15, 18, 22 of Algorithm 2, where $c_0^{\phi}, c_1^{\phi-1}, \ldots, c_\nu^{\phi-\nu}$ are as defined in (10). 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 $\boldsymbol{c}^0, \boldsymbol{c}^1, \ldots, \boldsymbol{c}^{n-1}$. However, this storage requirement is still linear in n.

6.2. 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 $\boldsymbol{c} = (1, 0, 1, 1, 0, 1, 1)$ used by Arıkan in [2].

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.

In Figure 12, we keep the list size constant at $L = 128$, but vary the constraint length ν. Note that setting $\nu = 0$ or $\nu = 1$ leads to degenerate cases. For $\nu = 0$, the matrix (10) reduces to the identity matrix and the 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 (10) is not time-varying and not random, with each row being a shift of the vector $\boldsymbol{c} = (1, 1)$. Thus the smallest nontrivial constraint length is $\nu = 2$, which allows a single bit of randomness per row in (10). Surprisingly, this suffices to closely match the performance of Arıkan's PAC code [2] with $\nu = 6$. As we increase the constraint length in (10) 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 (10) becomes a random nonsingular upper-triangular matrix. In Table 4, we compile (lower bounds on) the weight distribution for several typical realizations of the matrix in (10) which correspond to $\nu = 2, 6, 10$. These results corroborate the performance observed in simulations.

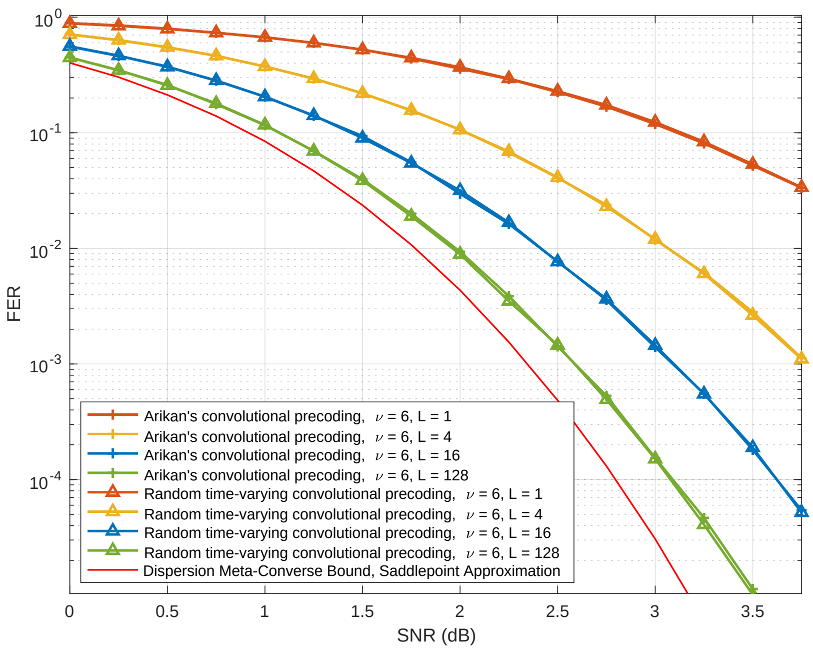

Figure 11. Performance of PAC codes for some specific realizations of random time-varying convolutional precoding with $\nu = 6$, as a function of the list size.

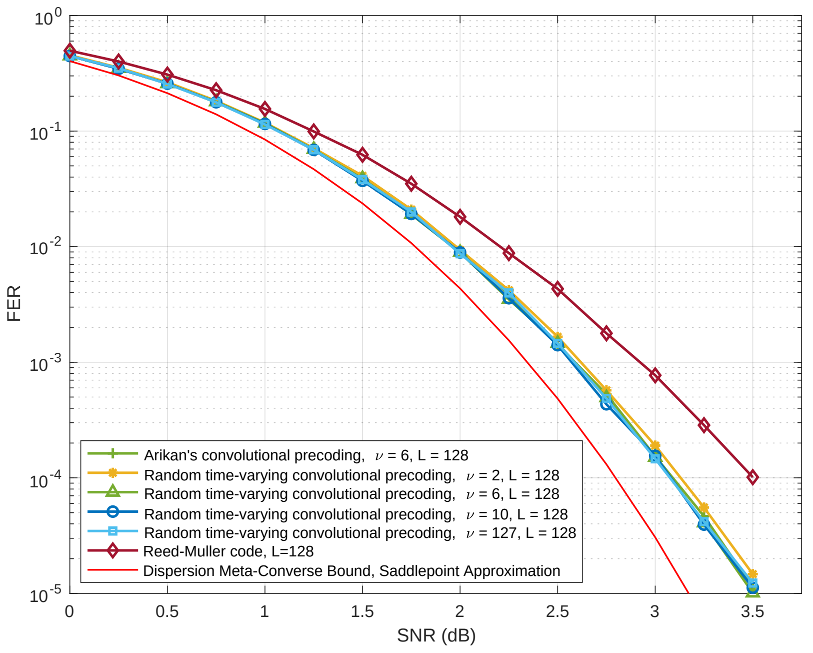

Figure 12. Performance of PAC codes for some specific realizations of random time-varying convolutional precoding for $L = 128$, as a function of the constraint length.

Table 4. Number of low-weight codewords in PAC codes for certain specific realizations of random time-varying convolutional 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	142,618	$>10^5$
Arıkan's PAC code with $\nu = 6$	0	3120	2696	95,828	$>10^5$
Random precoding with $\nu = 6$	0	2870	1526	88,250	$>10^5$
Random precoding with $\nu = 10$	0	2969	412	81,026	$>10^5$

7. 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 [60], 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.

Author Contributions: Conceptualization, H.Y., A.F. and A.V.; methodology, H.Y., A.F. and A.V.; software, H.Y. and A.F.; validation, H.Y., A.F. and A.V.; investigation, H.Y., A.F. and A.V.; writing, H.Y. and A.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Samsung Electronics under Sponsored Research Agreement titled *Variants of Polar Coding Technology*.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Acknowledgments: We are grateful to Erdal Arıkan for tremendous help with this work. Specifically, we are indebted to Erdal Arıkan for sharing the data and the source code of the sequential decoding program. We thank the referees for their constructive comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Arıkan, E. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. Inf. Theory* **2009**, *55*, 3051–3073. [CrossRef]
2. Arıkan, E. From sequential decoding to channel polarization and back again. *arXiv* **2019**, arXiv:1908.09594.
3. Polyanskiy, Y.; Poor, H.V.; Verdu, S. Channel coding rate in the finite blocklength regime. *IEEE Trans. Inf. Theory* **2010**, *56*, 2307–2359. [CrossRef]
4. Tal, I.; Vardy, A. List decoding of polar codes. *IEEE Trans. Inf. Theory* **2015**, *61*, 2213–2226. [CrossRef]
5. Li, B.; Shen, H.; Tse, D. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Commun. Lett.* **2012**, *16*, 2044–2047. [CrossRef]
6. Miloslavskaya, V.; Trifonov, P. Sequential decoding of polar codes. *IEEE Commun. Lett.* **2014**, *16*, 1127–1130. [CrossRef]
7. Niu, K.; Chen, K. CRC-aided decoding of polar codes. *IEEE Commun. Lett.* **2012**, *16*, 1668–1671. [CrossRef]
8. 3GPP Technical Specification Group Radio Access Network,
"Multiplexing and channel coding" Release 16, 3GPP TS 38.212 V16.3.0. November 2020. Available online: https://www.etsi.org/deliver/etsi_ts/138200_138299/138212/16.03.00_60/ts_138212v160300p.pdf (accessed on 27 June 2021).
9. Erseghe, T. Coding in the finite-blocklength regime: Bounds based on Laplace integrals and their asymptotic approximations. *IEEE Trans. Inf. Theory* **2016**, *62*, 6854–6883. [CrossRef]
10. Fano, R. A heuristic discussion of probabilistic decoding. *IEEE Trans. Inf. Theory* **1963**, *9*, 64–74. [CrossRef]

11. Gallager, R.G. *Information Theory and Reliable Communication*; Wiley: New York, NY, USA, 1968.
12. Trifonov, P. A score function for sequential decoding of polar codes. In Proceedings of the IEEE International Symposium on Information Theory, Vail, CO, USA, 21–26 June 2018; pp. 1470–1474.
13. Fazeli, A.; Hassani, H.; Mondelli, M.; Vardy, A. Binary linear codes with optimal scaling: Polar codes with large kernels. *IEEE Trans. Inf. Theory* **2020**, doi:10.1109/TIT.2020.3038806. [CrossRef]
14. Fazeli, A.; Vardy, A. On the scaling exponent of binary polarization kernels. In Proceedings of the Allerton Conference Communication, Control, and Computing, Monticello, IL, USA, 30 September–3 October 2014; pp. 797–804.
15. Korada, S.B.; Şaşoğlu, E.; Urbanke, R. Polar codes: Characterization of exponent, bounds, and constructions. *IEEE Trans. Inf. Theory* **2010**, *56*, 6253–6264. [CrossRef]
16. Moskovskaya, E.; Trifonov, P. Design of BCH polarization kernels with reduced processing complexity. *IEEE Commun. Lett.* **2020**, *24*, 1383–1386. [CrossRef]
17. Trifonov, P. On construction of polar subcodes with large kernels. In Proceedings of the IEEE International Symposium on Information Theory, Paris, France, 7–12 July 2019; pp. 1932–1936.
18. Trifonov, P. Trellis-based decoding techniques for polar codes with large kernels. In Proceedings of the IEEE Information Theory Workshop, Visby, Sweden, 25–28 August 2019; pp. 1–5.
19. Trofimiuk, G.; Trifonov, P. Efficient decoding of polar codes with some 16×16 kernels. In Proceedings of the IEEE Information Theory Workshop, Guangzhou, China, 25–29 November 2018; pp. 11–15.
20. Trofimiuk, G.; Trifonov, P. Reduced complexity window processing of binary polarization kernels. In Proceedings of the IEEE International Symposium on Information Theory, Paris, France, 7–12 July 2019; pp. 1412–1416.
21. Yao, H.; Fazeli, A.; Vardy, A. Explicit polar codes with small scaling exponent. In Proceedings of the IEEE International Symposium on Information Theory, Paris, France, 7–12 July 2019; pp. 1757–1761.
22. Morozov, R.; Trifonov, P. Successive and two-stage systematic encoding of polar subcodes. *IEEE Wireless Commun. Lett.* **2019**, *8*, 877–880. [CrossRef]
23. Trifonov, P. Star polar subcodes. In Proceedings of the IEEE Wireless Communications and Networking Conference, San Francisco, CA, USA, 17–21 March 2017; pp. 1–6.
24. Trifonov, P. Randomized polar subcodes with optimized error coefficient. *IEEE Trans. Commun.* **2020**, *68*, 6714–6722. [CrossRef]
25. Trifonov, P.; Miloslavskaya, V. Polar codes with dynamic frozen symbols and their decoding by directed search. In Proceedings of the IEEE Information Theory Workshop, Sevilla, Spain, 9–13 September 2013; pp. 1–5.
26. Trifonov, P.; Miloslavskaya, V. Polar subcodes. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 254–266. [CrossRef]
27. Trifonov, P.; Trofimiuk, G. A randomized construction of polar subcodes. In Proceedings of the IEEE International Symposium on Information Theory, Aachen, Germany, 25–30 June 2017; pp. 1863–1867.
28. Ferris, A.J.; Poulin, D. Branching MERA codes: A natural extension of polar codes. *arXiv* **2013**, arXiv:1312.4575.
29. Ferris, A.J.; Hirche, D.C.; Poulin, D. Convolutional polar codes. *arXiv* **2017**, arXiv:1704.00715.
30. Morozov, R. Convolutional polar kernels. *IEEE Trans. Commun.* **2020**, *68*, 7352–7361. [CrossRef]
31. Morozov, R.; Trifonov, P. On distance properties of convolutional polar codes. *IEEE Trans. Commun.* **2019**, *67*, 4585–4592. [CrossRef]
32. Abbe, E.; Ye, M. Reed-Muller codes polarize. *IEEE Trans. Inf. Theory* **2020**, *66*, 7311–7332. [CrossRef]
33. Li, B.; Shen, H.; Tse, D. RM-polar codes. *arXiv* **2014**, arXiv:1407.5483.
34. Mondelli, M.; Hassani, S.H.; Urbanke, R.L. From polar to Reed–Muller codes: A technique to improve the finite-length performance. *IEEE Trans. Commun.* **2020**, *62*, 3084–3091. [CrossRef]
35. Ye, M.; Abbe, E. Recursive projection-aggregation decoding of Reed–Muller codes. *arXiv*, **2019**, arXiv:1902.01470.
36. Coşkun, M.C.; Neu, J.; Pfister, H.D. Successive cancellation inactivation decoding for modified Reed–Muller and eBCH codes. In Proceedings of the IEEE International Symposium on Information Theory, Los Angeles, CA, USA, 21–26 June 2020; pp. 437–442.
37. Miloslavskaya, V.; Vucetic, B. Design of short polar codes for SCL decoding. *IEEE Trans. Commun.* **2021**, *68*, 6657–6668. [CrossRef]
38. Yuan, P.; Prinz, T.; Böcherer, G.; Iscan, O.; Boehnke, R.; Xu, W. Polar code construction for list decoding. In Proceedings of the 12th International ITG Conference on Systems, Communications and Coding, Rostock, Germany, 11–14 February 2019; pp. 1–6.
39. Fazeli, A.; Tian, K.; Vardy, A. Viterbi-aided successive-cancellation decoding of polar codes. In Proceedings of the IEEE Global Communications Conference, Singapore, 4–8 December 2017; pp. 1–6.
40. Fazeli, A.; Vardy, A.; Yao, H. Convolutional decoding of polar codes. In Proceedings of the IEEE International Symposium on Information Theory, Paris, France, 7–12 July 2019; pp. 1397–1401.
41. Arıkan, E. Systematic encoding and shortening of PAC Codes. *Entropy* **2020**, *22*, 1301–1304. [CrossRef]
42. Li, B.; Zhang, H.; Gu, J. On pre-transformed polar codes. *arXiv* **2019**, arXiv:1912.06359.
43. Mishra, S.K.; Kim, K.C. Selectively precoded polar codes. *arXiv* **2020**, arXiv:2011.04930.
44. Moradi, M.; Mozammel, A.; Qin, K.; Arıkan, E. Performance and complexity of sequential decoding of PAC codes. *arXiv* **2020**, arXiv:2012.04990v1.
45. Mozammel, A. Hardware implementation of Fano decoder for PAC codes. *arXiv* **2020**, arXiv:2011.09819.
46. Tonnellier, T.; Gross, W.J. On systematic polarization-adjusted convolutional (PAC) codes. *arXiv* **2020**, arXiv:2011.03177.

47. Wang, L.; Jiang, M.; Zhao, C.; Li, Z. Genetic optimization of short block-length PAC codes for high capacity PHz communications. In Proceedings of the International Conference on Optoelectronic and Microelectronic Technology and Application, Nanjing, China, 17–19 October 2020; Volume 11617.
48. Rowshan, M.; Burg, A.; Viterbo, E. Polarization-adjusted convolutional (PAC) codes: Fano decoding vs. list decoding. *arXiv* **2020**, arXiv:2002.06805v1.
49. Rowshan, M.; Viterbo, E. List Viterbi decoding of PAC codes. *arXiv* **2020**, arXiv:2007.05353.
50. Yao, H.; Fazeli, A.; Vardy, A. List decoding of Arıkan's PAC codes. In Proceedings of the IEEE International Symposium on Information Theory, Los Angeles, CA, USA, 21–26 June 2020; pp. 443–448.
51. Vazquez-Vilar, G.; Fabregas, A.G.; Koch, T.; Lancho, A. Saddlepoint approximation of the error probability of binary hypothesis testing. In Proceedings of the IEEE International Symposium on Information Theory, Vail, CO, USA, 17–22 June 2018; pp. 2306–2310.
52. Coşkun, M.C.; Durisi, G.; Jerkovits, T.; Liva, G.; Ryan, W.; Stein, B.; Steiner, F. Efficient error-correcting codes in the short blocklength regime. *Phys. Commun.* **2019**, *34*, 66–79. [CrossRef]
53. Goldin, D.; Burshtein, D. Performance bounds of concatenated polar coding schemes. *IEEE Trans. Inf. Theory* **2019**, *65*, 7131–7148. [CrossRef]
54. Sarkis, G.; Giard, P.; Vardy, A.; Thibeault, C; Gross, W.J. Increasing the speed of polar list decoders. In Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS), Belfast, UK, 20–22 October 2014; pp. 1–6.
55. Sarkis, G.; Giard, P.; Vardy, A.; Thibeault, C.; Gross, W.J. Fast list decoders for polar codes. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 318–328. [CrossRef]
56. Balatsoukas-Stimming, A. *Private Communication*, August 2019.
57. Sugino, M.; Ienaga, Y.; Tokura, N.; Kasami, T. Weight distribution of (128,64) Reed–Muller code. *IEEE Trans. Inf. Theory* **1971**, *17*, 627–628. [CrossRef]
58. Yao, H.; Fazeli, A.; Vardy, A. A deterministic algorithm for computing the weight distribution of polar codes. *arXiv* **2021**, arXiv:2102.07362.
59. Sason, I.; Shamai, S. Performance analysis of linear codes under maximum-likelihood decoding: A tutorial. *Found. Trends Commun. Inf. Theory* **2006**, *3*, 1–225. [CrossRef]
60. Zhu, H.; Cao, Z.; Zhao, Y.; Li, D.; Yang, Y.; Wang, Y.; Guo, Z. Fast list decoders for polarization-adjusted convolutional (PAC) codes. *arXiv* **2020**, arXiv:20122.09425.