List Decoding of Arıkan's PAC Codes

Hanwen Yao, Arman Fazeli, and Alexander Vardy University of California San Diego, La Jolla, CA 92093, USA {hwyao,avardy,afazelic}@ucsd.edu

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 block lengths under standard successive cancellation decoding is far from optimal. A well-known way to improve the performance of polar codes at short block lengths 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. Last year, Arıkan presented a new polar coding scheme, which he called polarization-adjusted convolutional (PAC) codes. Such PAC codes provide another dramatic improvement in performance as compared to CRC-aided list decoding. These codes are based primarily upon the following main ideas: replacing CRC precoding with convolutional precoding (under appropriate rate profiling) and replacing list decoding by sequential decoding. Arikan's simulation results show that PAC codes, resulting from the combination of these ideas, are quite 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, $L \ge 128$). 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 under ML decoding. These results indicate that PAC codes are superior to polar codes and Reed-Muller codes, and suggest that the goal of rateprofiling may be to optimize the weight distribution at low weights.

Index Terms—coding theory, polar codes, convolutional codes, successive-cancellation list decoding, sequential decoding

I. 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 adopt this strategy herein. We make no attempt to optimize these codes; rather, our goal is to follow Arıkan [2]



Figure 1. Performance of PAC codes versus polar codes

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

As shown in [20] 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 [11]–[13], has largely remained the state of the art in polar coding since it was first introduced in [20].

In his Shannon Lecture at the ISIT in 2019, Erdal Arıkan presented a significant breakthrough in polar coding, which 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. 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

Arıkan's PAC codes [2] are based primarily upon the following two innovations: replacing CRC precoding with *convolutional precoding* (under appropriate rate-profiling, to be discussed later) and replacing list decoding by *sequential decoding*. The

To be presented in part at the IEEE International Symposium on Information Theory in June 2020 (submitted for review January 15, 2020).



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

encoding and decoding of PAC codes are shown schematically in Figure 2, which is reproduced from [2].

Referring to Figure 2, let's 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 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, 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 V.

The main operation on the decoder side is sequential decoding. Specifically, Arıkan employs Fano decoding 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. 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 [11] 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 1 and Figure 9. These results suggest that the goal of rate-profiling may be to optimize the weight distribution at low weights.

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 [20], sequential decoding [12], and stack decoding [13], 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 studied by Trifonov and Miloslavskaya in [21], although the dynamic freezing patters in [21] and [2] are very different. Prior to Arıkan's paper [2], convolutional precoding of polar codes was proposed in [7] and later studied in [8]. Finally, the work of [15], which considers Fano and list decoding of PAC codes, is independent from and contemporaneous with our results herein¹.

D. Paper Outline

The rest of this paper is organized as follows. We begin with an overview on Arıkan'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. We conclude with a brief discussion in Section VI.

II. OVERVIEW OF ARIKAN'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 $m \ge 1$. As shown in Figure 2, the encoding process for an (n, k) PAC code consists of the following three steps: rate-profiling, convolutional precod-

¹The work of Rowshan, Burg, and Viterbo [15], was posted on arxiv.org in February 2020, while our work was submitted for review in January 2020.

ing, and polar encoding. In the first step, the *k* data (information) bits of the data vector *d* are embedded into a data-carrier vector *v* of length *n*, at *k* positions specified by an index set $\mathcal{A} \subseteq \{0, 1, ..., n-1\}$ with $|\mathcal{A}| = k$. The remaining n - k positions in *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 Ais 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 rateprofiling, \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 $\{W_i : i \in A\}$ 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 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 {wt(i) : $i \in A$ } consists of the k largest values among $wt(0), wt(1), \ldots, wt(n-1)$. Notably, without convolutional precoding, this choice of A generates Reed-Muller codes (as subcodes of a rate-1 polar code).

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

$$u_0 = c_0 v_0,$$

$$u_1 = c_0 v_1 + c_1 v_0,$$

$$u_2 = c_0 v_2 + c_1 v_1 + c_2 v_0,$$

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

$$u_{i} = \sum_{j=0}^{\nu} c_{j} v_{i-j}$$
(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 $u = v\mathbf{T}$, where **T** is the upper-triangular Toeplitz matrix:

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

In the third step, the vector u is finally encoded by a conventional polar encoder as the codeword $x = uP_m$. 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, 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 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 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 datacarrier vector 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 [6] and [9]. 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 [2].

A. PAC Codes as Polar Codes with Dynamic Freezing

To achieve efficient list decoding of PAC codes, we use the listdecoding algorithm developed in [20]. The complexity of this algorithm is $O(Ln \log n)$, where L is the list size. However, the algorithm of [20] 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 [21], and later studied by the same authors in [22] and [23]. 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 $i \in \mathcal{F}$. Arıkan showed in [1] that on symmetric channels, this does not affect the performance. What matters is that the frozen bits are fixed and, therefore, known *a priori* to the decoder. In [21], 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

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

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

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.

Algorithm 1: List Decoder for PAC Codes	Algorithm 2: continuePaths_Unfzn (PAC version)			
Input: The received vector \boldsymbol{y} , the list size L , the generator $\boldsymbol{c} = (c_0, c_1, \dots, c_v)$ for the convolutional precoder as global	<pre>Input: phase φ 1 ··· lines 1-18 of Algorithm 13 in [20] // Continue relevant paths</pre>			
Output: Decoded codeword x	2 for $\ell = 0, 1, \dots, L-1$ do			
// Initialization	3 if $contForks[\ell][0] = false and$			
1 ··· lines 2–5 of Algorithm 12 in [20] 2 shiftPagisters $(1, 1)$ power 2 D array of size $I \times (1, 1)$	$controrks[\ell][1] = $ false then			
2 sintregisters \leftarrow new 2-D analy of size $L \times (\ell + 1)$ 3 for $\ell = 0.1$ $L = 1$ do				
4 shiftRegister[ℓ] = (0,0,,0)	5 $C_m \leftarrow \text{getArrayPointer}(C(m, \ell))$			
// Main Loon	6 left-shift shiftRegister[ℓ] by one, with the rightmost			
5 for $\alpha = 0.1$ $n = 1.40$	7 $(72 + 72 + 1 + 72 + 1) \leftarrow \text{shiftRegister}[\ell]$			
$6 \mid \text{recursivelyCalcP}(m, \varphi)$	$\mathbf{if } contForks[\ell][0] = \mathbf{true} and contForks[\ell][1] = \mathbf{true}$			
7 if u_{ω} is frozen then	then			
8 $\vec{for} \ \ell = 0, 1,, L - 1 \ do$	9 $C_m[0][\varphi \mod 2] \leftarrow \sum_{i=0}^{\nu} c_i v_{\varphi-i}$			
9 if active Path $[\ell]$ = false then	10 $\ell' \leftarrow \text{clonePath}(\ell)$			
10 continue	11 shiftRegister[ℓ'] \leftarrow shiftRegister[ℓ]			
11 left-shift shiftRegister[ℓ] by one,	12 flip the rightmost bit of shiftRegister[ℓ']			
with the rightmost position set to 0	13 $C_m \leftarrow \text{getArrayPointer}_C(m, \ell')$			
12 $C_m \leftarrow \text{getArrayPointerC}(m, \ell)$	14 $(v'_{\varphi-\nu}, v_{\varphi-\nu+1}, \dots, v'_{\varphi}) \leftarrow \text{shiftRegister}[\ell']$			
13 $(v_{\varphi-\nu}, v_{\varphi-\nu+1}, \dots, v_{\varphi}) \leftarrow \text{shiftRegister}[\ell]$	15 $C_m[0][\varphi \mod 2] \leftarrow \sum_{j=0}^{\nu} c_j v'_{\varphi-j}$			
// Set the frozen bit				
14 $[C_m[0][\varphi \mod 2] \leftarrow \sum_{j=0}^{n} c_j v_{\varphi-j}]$	17 If contForks $[\ell][0]$ = true then 18 $if \Sigma^{V} = c \pi = 1$ then			
15 else	18 If $\sum_{j=0} c_j c_{\phi-j} = 1$ then the dimetric of shift Desister [ℓ]			
16 continuePaths_Unfzn(φ)	If the right host of sint Register $[t]$			
17 if $\varphi \mod 2 = 1$ then	20 set $C_m[0][\varphi \mod 2] \leftarrow 0$			
18 recursivelyUpdateC (m, φ)	21 else $if \nabla^{V} = c \pi = 0$ then			
$ \sqsubseteq$	$\sum_{j=0}^{22} c_j v_{\varphi-j} = 0 \text{ then}$			
$19 \cdots$ lines 17–24 of Algorithm 12 in [20]	23 inp the rightmost bit of shiftKegister[ℓ]			
20 return $\hat{x} = (C_0[\beta][0])_{\beta=0}^{n-1}$	24 $ [$			
(0)[P][0]/B=0				

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 II, for i = 0, 1, ..., n-1, let u_i and v_i denote the vectors (u_0, u_1, \ldots, u_i) and (v_0, v_1, \ldots, v_i) , respectively. Further, let $T_{i,i}$ denote the submatrix of the Toepliz matrix 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 $u_i = v_i \mathbf{T}_{i,i}$ 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 $v_i = u_i \mathbf{T}_{ii}^{-1}$ 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

$$\boldsymbol{u}_{i} = \boldsymbol{v}_{i-1} \mathbf{T}_{i-1,i} = \left(\boldsymbol{u}_{i-1} \mathbf{T}_{i-1,i-1}^{-1} \right) \mathbf{T}_{i-1,i}$$
(5)

In particular, this means that the last bit u_i of the vector 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 $(0, \ldots, 0, c_{\nu}, c_{\nu-1}, \ldots, c_1)^t$ represents the last column of the matrix $\mathbf{T}_{i-1,i}$. Clearly, the above is a special case of dynamic freezing in (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 $i \in A$, 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 (5). Instead, we successively compute the vector $\hat{v} = (\hat{v}_0, \hat{v}_1, \dots, \hat{v}_{n-1})$ as follows. If $i \in \mathcal{A}^c$, set $\hat{v}_i = 0$. Otherwise, set

$$\widehat{v}_i = \widehat{u}_i - \sum_{j=1}^{\nu} c_j \widehat{v}_{i-j} \tag{6}$$

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 $\hat{v} = (\hat{v}_0, \hat{v}_1, \dots, \hat{v}_{n-1})$.

B. List Decoding of PAC codes

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

There are, however, several important differences. For example, for conventional polar codes, whenever 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 the 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 [20]. In addition, for a list of size L, we introduce L auxiliary shift registers — one for each path. Each shift register stores the last ν bits of the vector $\hat{v} = (\hat{v}_0, \hat{v}_1, \dots, \hat{v}_{n-1})$, 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 [20], 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 Arıkan in [2]. We are grateful to Erdal Arıkan for sharing the details of his decoding algorithm. We do not disclose these details here, instead referring the reader to [2].

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 Arıkan's 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 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 until we reach a point of diminishing returns. 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



Figure 3. Performance of PAC codes under list decoding

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 [2]. 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 $L \ge 64$) 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 [14]. Our curve coincides with those given in [4, Figure 1] and [10, Figure 6]. Note that a more accurate bound can be derived using the methods of Erseghe [5], but this is not critical for our purposes. It is clear from Figures





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

1 and 3 that the performance of the (128, 64) PAC code, under both sequential decoding and list decoding with $L \ge 128$, 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 [16], [17], 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 [3].

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 $u = (u_0, u_1, \ldots, u_{n-1})$ 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 5. Sequential decoding vs. list decoding complexity comparison: Number of nodes visited in the polar search tree per codeword



Figure 6. Sequential decoding vs. list decoding complexity comparison: 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 $i \in A$ 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 $L(k + 2 - \log_2 L) - 1 = O(kL)$. If *L* is not a power of 2, this counting argument readily generalizes, and the number of decision nodes visited is given by

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

As another 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 independent of SNR, resulting in flat curves. For sequential decoding, we 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 complex-ity/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 Arıkan's (128, 64) PAC code as well.



Figure 7. Performance of the PAC code under ML decoding

Figure 7 shows a bound on the frame error-rate of ML decoding obtained in our simulations. This is a *lower bound*, in the sense that the actual simulated performance of ML decoding could be only 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 pathmetric (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 [20] for polar codes, except that [20] 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 Arıkan's PAC codes are so good at short blocklengths: they can be efficiently decoded with near-ML fidelity.

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 [11] to estimate the number of low-weight codewords in this code.

Consider the following experiment devised in [11]. 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 lowweight 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, $A_{16} \ge 3120$, $A_{18} \ge 2696$, and $A_{20} \ge 95828$ (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.



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

For comparison, 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. The results are compiled in Table 1 below.

	A_8	A ₁₂	A ₁₆	A ₁₈	A ₂₀	A ₂₂
Polar code	4	0	68856	0	$> 10^5$	-
Polar code, CRC8	20	173	≥ 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 [19]). 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). 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 [18], 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.



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



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

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 establishes another reason for the remarkable performance of this code.

VI. 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.

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. We hope our work stimulates 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 convolutional precoder? We hope our results in Section V 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

- E. Arıkan, "Channel polarization: A method for constructing capacityachieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, pp. 3051–3073, 2009.
- [2] E. Arıkan, "From sequential decoding to channel polarization and back again", preprint available as arXiv:1908.09594, September 2019.
- [3] A. Balatsoukas-Stimming, private communication, August 2019.
- [4] 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.
- [5] 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.

- [6] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans*actions on Information Theory, vol. 9, pp. 64–74, April 1963.
- [7] 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.
- [8] 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.
- [9] R.G. Gallager, Information Theory and Reliable Communication. New York: Wiley, 1968.
- [10] D. Goldin and D. Burshtein, "Performance bounds of concatenated polar coding schemes," *IEEE Transactions on Information Theory*, vol. 65, pp. 7131–7148, November 2019.
- [11] 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.
- [12] V. Miloslavskaya and P. Trifonov, "Sequential decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 7, pp. 1127–1130, July 2014.
- [13] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Com*mun. Letters, vol.16, no. 10, pp. 1668–1671, October 2012.
- [14] Y. Polyanskiy, H.V. Poor, and S. Verdu, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, pp. 2307–2359, May 2010.
- [15] 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.
- [16] G. Sarkis, P. Giard, A.Vardy, C. Thibeault, 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.
- [17] G. Sarkis, P. Giard, A.Vardy, C. Thibeault, and W.J. Gross, "Fast list decoders for polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 318–328, February 2016.
- [18] 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.
- [19] M. Sugino, Y. Ienaga, Y. Tokura, and T. Kasami, "Weight distribution of (128, 64) Reed-Muller code," *IEEE Transactions on Information The*ory, vol. 17, pp. 627–628, September 1971.
- [20] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, pp. 2213–2226, May 2015.
- [21] 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.
- [22] P. Trifonov and V. Miloslavskaya, "Polar subcodes," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 254–266, February 2016.
- [23] 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.