Permuted Successive Cancellation Decoding for Polar Codes

Sarit Buzaglo, **Arman Fazeli**, **Paul H. Siegel**, **Veeresh Taranalli**, and **Alexander Vardy**
University of California San Diego, La Jolla, CA 92093, USA
{*sbuzaglo, afazelic, psiegel, vtaranalli, avardy*}*@ucsd.edu*

***Abstract*—Defined through a certain 2×2 matrix called *Arikan's kernel*, polar codes are known to achieve the symmetric capacity of binary-input discrete memoryless channels under the *successive cancellation* (SC) decoder. Yet, for short block-lengths, polar codes fail to deliver a compelling performance under the low complexity SC decoding scheme. Recent studies provide evidence for improved performance when Arikan's kernel is replaced with larger kernels that have smaller *scaling exponents*. However, for $\ell \times \ell$ kernels the time complexity of the SC decoding increases by a factor of 2^ℓ.**

In this paper we study a special type of kernels called *permuted kernels*. The advantage of these kernels is that the SC decoder for the corresponding polar codes can be viewed as a permuted version of the SC decoder for the conventional polar codes that are defined through Arikan's kernel. This *permuted successive cancellation* (PSC) decoder outputs its decisions on the input bits according to a permuted order of their indices. We introduce an efficient PSC decoding algorithm and show simulations for two 16×16 permuted kernels that have better scaling exponents than Arikan's kernel.

***Index Terms*—Large kernels, polar codes, successive cancellation decoding, scaling exponent.**

I. Introduction

Introduced by Arikan [1], polar codes are the first codes that were proved to achieve the symmetric capacity of a binary-input discrete memoryless channel (B-DMC) $\mathcal{W}$. Polar codes can be viewed as part of a much larger family of codes that are generated according to the 2×2 matrix

$$F_2 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

These length $n = 2^m$ codes are cosets of a linear subspace that is spanned by some k rows of $F_2^{\otimes m}$, the mth Kronecker power of F_2. One important example of such codes are Reed-Muller codes, for which the spanning k rows correspond to the k rows of $F_2^{\otimes m}$ that have maximum Hamming weight. The genius of Arikan's construction lies in the fact that the selection of the k spanning rows depends on the channel. Each row of $F_2^{\otimes m}$ is associated with a synthesized channel called a *bit channel*, which is defined through F_2 and the channel $\mathcal{W}$. The selected k rows correspond to the k least noisy bit channels, where the noisiness of the channel is measured by its Bhattacharyya parameter (see [1] for more information on the Bhattacharyya parameter). A remarkable property of F_2 is that as n grows, the bit channels exhibit a polarization phenomenon: some of the bit channels become very noisy, while the rest of the bit channels become almost noiseless. Moreover, the fraction of channels that are almost noiseless approaches the symmetric capacity of the channel, $I(\mathcal{W})$. As a result, for every $R < I(\mathcal{W})$ and for sufficiently large n there exists a rate R polar code for which the frame error rate (FER) under a *successive cancellation* (SC) decoder is $O(n^{-\frac{1}{2+\epsilon}})$, where $\epsilon > 0$ is arbitrarily small.

Despite their impressive asymptotic behavior, empirical results indicate less impressive performance of the SC decoder for polar codes of short block-lengths, e.g., compared to LDPC codes [13]. To improve performance, different decoders were suggested, e.g., belief propagation decoding [2], [6], [9], improved SC decoding [5], and list successive cancellation decoding [13]. Combined with cyclic redundancy check (CRC) pre-coding, the list successive cancellation decoder provides the best performance for polar codes to date. Meanwhile, a series of works pursued the same goal by replacing F_2 with larger matrices called *kernels* that may induce a faster polarization of the bit channels [7], [8], [10], [11], [12], [15].

To estimate how polarizing a kernel is, two figures of merit were proposed. The first is the *error exponent* [3], [12] which quantifies the exponential decay of the error probability with respect to the block-length n, for a fixed rate R. The second is the *scaling exponent* [7], [10], [16] which reflects the dependence between the length of the code and its rate, for a fixed probability of error P_e. More precisely, a scaling exponent μ is a constant that depends only on the kernel K and the channel $\mathcal{W}$ for which $n = \Theta\left((I(\mathcal{W}) - R)^{-\mu}\right)$, where the sum of the Bhattacharyya parameters of the nR least noisy bit channels is at most P_e. No method is known to compute the scaling exponent for general channels. In fact, it is not even clear whether or not the scaling exponent exists. Hassani *et. al.* [10] introduced a numerical method to compute the scaling exponent of polar codes for the binary erasure channel (BEC) and found that $\mu = 3.627$, where the kernel is F_2. Fazeli and Vardy [7] presented an 8×8 kernel K_8 with $\mu = 3.577$ and showed that this is optimal for kernels of size $\ell \leq 8$. They also suggested a heuristic construction through which they found a 16×16 kernel K_{16} with $\mu = 3.356$. However, the time complexity of a SC decoding of polar codes with $\ell \times \ell$ kernels scales as $2^\ell n \log_\ell n$. In [11] a method to reduce the time complexity of a SC decoding of polar codes with large kernels was suggested. This method was shown to be efficient for certain $\ell \times \ell$ kernels, with $\ell \leq 16$.

In this paper we suggest a different approach towards polar codes with high performance and efficient SC decoding. In our approach we consider a special type of kernels called *permuted kernels*. These kernels are formed by permuting the rows of $F_2^{\otimes \ell}$. One example of a permuted kernel is the kernel K_8 from [7]. On the other hand, the kernel K_{16} defined in [7] is not a permuted kernel. While a successive cancellation decoder for a polar code with the kernel F_2

978-1-5090-4096-4/17/$31.00 ©2017 IEEE

and dimension k decides on the n input bits $u_0u_1\dots u_{n-1}$ ($n-k$ of them are known to the decoder) one after the other according to the sequential order from 0 to $n-1$, a SC decoder for polar codes with permuted kernels decides on the input bits according to a permuted order of their indices. Therefore, we call the SC decoder for a polar code with a permuted kernel a *permuted successive cancellation* (PSC) decoder. A PSC decoding algorithm was presented in [4], however in this work we further exploit the connection between permuted kernels and $F_2^{\otimes\ell}$ and introduce a more efficient implementation of the PSC decoder, in terms of both time and space complexity. Due to page limitations, we only describe our PSC decoding algorithm for length-ℓ polar codes. A more detailed presentation of the algorithm will be given in the full version of this paper. We also propose two new 16×16 permuted kernels and show simulation results for their performance.

The rest of this paper is organized as follows. In Section II, we present notation and definitions that are used throughout the paper and review the basic concepts of polar codes. We also discuss the connection between the bit channels of a permuted kernel and the bit channels of $F_2^{\otimes\ell}$. Our PSC decoding algorithm for block-length ℓ is presented in Section III. Time and space complexity of the algorithm are discussed in Section IV along with some simulation results.

II. Preliminaries

In this section we introduce notation and definitions used throughout the paper and review the basic concepts for polar codes.

For a positive integer n, denote by $[n]$ the set of n integers $\{0,1,\dots,n-1\}$. For a positive integer ℓ and for all $r\in[\ell]$, denote by $[n]_r$ the set of all elements in $[n]$ that are equal to r modulo ℓ, where ℓ should be clear from the context. A binary vector of length n is denoted by $u_0^{n-1}=u_0u_1\dots u_{n-1}$. For $\mathcal{A}\subseteq[n]$, denote by $u_{\mathcal{A}}$ the subvector of u_0^{n-1} that is specified according to indices from $\mathcal{A}$. In particular, if ℓ divides n and $r\in[\ell]$ then $u_{[n]_r}=u_ru_{\ell+r}\dots u_{n-\ell+r}$. All operations on vectors and matrices in this paper are carried out over the field $GF(2)$. The componentwise addition modulo-2 of two binary vectors u_0^{n-1} and v_0^{n-1} is denoted by $u_0^{n-1}\oplus v_0^{n-1}$. For an $\ell\times\ell$ matrix K, denote by $K^{\otimes m}$ the mth Kronecker power of K.

Throughout this paper $\mathcal{W}:\mathcal{X}\to\mathcal{Y}$ is a generic B-DMC with input alphabet $\mathcal{X}=\{0,1\}$, output alphabet $\mathcal{Y}$, and *transition probabilities* $\mathcal{W}(y|x)$, where for all $x\in\mathcal{X}$ and $y\in\mathcal{Y}$, $\mathcal{W}(y|x)$ is the conditional probability that the channel output is y given that the transmitted input is x.

For a positive integer n, denote by $\mathcal{W}^n:\mathcal{X}^n\to\mathcal{Y}^n$ the channel that corresponds to transmission over n independent copies of $\mathcal{W}$.

A *kernel* K is an invertible $\ell\times\ell$ matrix. Let $n=\ell^m$ and let R_n be the permutation matrix for which $u_0^{n-1}R_n=u_{[n]_0}u_{[n]_1}\dots u_{[n]_{\ell-1}}$, for all $u_0^{n-1}\in\mathcal{X}^n$. For an $\ell\times\ell$ kernel K, define the matrix $G_{m,K}$ recursively by $G_{1,K}\stackrel{\text{def}}{=}K$ and $G_{m,K}\stackrel{\text{def}}{=}(I_{n/\ell}\otimes K)R_n(I_\ell\otimes G_{m-1,K})$. For a set $\mathcal{A}\subset[n]$ of size k, and a vector f_0^{n-k-1}, let $\mathcal{C}$ be the code that encodes a length n input vector u_0^{n-1}, for which $u_{[n]\setminus\mathcal{A}}=f_0^{n-k-1}$, to the codeword $x_0^{n-1}=u_0^{n-1}G_{m,K}$. If $i\in[n]\setminus\mathcal{A}$, then u_i is called a *frozen bit*. For all $i\in[n]$, the *ith bit channel* with respect to $G_{m,K}$, $\mathcal{W}_{m,K}^{(i)}:\mathcal{X}\to\mathcal{Y}^n\times\mathcal{X}^i$, is defined by the transition probabilities

$$\mathcal{W}_{m,K}^{(i)}(y_0^{n-1},u_0^{i-1}|u_i)\stackrel{\text{def}}{=}\sum_{u_{i+1}^{n-1}\in\mathcal{X}^{n-i-1}}\frac{1}{2^{n-1}}\mathcal{W}^n(y_0^{n-1}|u_0^{n-1}G_{m,K}).$$

A polar code $\mathcal{C}$ of length m and with kernel K is defined by setting $\mathcal{A}$ to be the set of k indices corresponding to the bit-channels with the lowest Bhattacharyya parameters[1]. A *successive cancellation* (SC) decoder for $\mathcal{C}$ outputs a *decision vector* $\hat{u}_0^{n-1}$ in n steps, where at the ith step the decoder decides on the value of $\hat{u}_i$ according to the following rule. If $i\in[n]\setminus\mathcal{A}$ then $\hat{u}_i$ is set to the value of the frozen bit u_i. Otherwise, the decoder calculates the pair of transition probabilities

$$\mathcal{W}_{m,K}^{(i)}(y_0^{n-1},\hat{u}_0^{i-1}|u_i=0),\ \mathcal{W}_{m,K}^{(i)}(y_0^{n-1},\hat{u}_0^{i-1}|u_i=1). \tag{1}$$

and sets $\hat{u}_\varphi$ to the more likely value according to these probabilities. Notice that, in general, the complexity of the SC decoder may be exponential in n, since the calculation of the transition probabilities requires a summation of 2^{n-i-1} terms.

The recursive structure of the matrix $G_{m,K}$ induces recursive formulas for the bit channels as follows.

Lemma 1. *Let $K=(K_{r,s})$ be an $\ell\times\ell$ kernel. For all $i\in[n]$, if $i=\varphi\ell+j$ for some $\varphi\in[n/\ell]$ and $j\in[\ell]$ then*

$$\mathcal{W}_{m,K}^{(i)}(y_0^{n-1},u_0^{i-1}|u_i)=\frac{1}{2^{\ell-1}}\sum_{u_{\varphi\ell+j+1}^{(\varphi+1)\ell-1}}\prod_{s=0}^{\ell-1}\mathcal{W}_{m-1,K}^{(\varphi)}(y_{s\ell^{n/\ell-1}}^{(s+1)n/\ell-1},T^{(s)}(u_{[\varphi\ell]})|\oplus_{r=0}^{\ell-1}K_{r,s}\cdot u_{\varphi\ell+r}),$$

where $T^{(s)}(u_{[\varphi\ell]})\stackrel{\text{def}}{=}\oplus_{r=0}^{\ell-1}K_{r,s}\cdot u_{[\varphi\ell]_r}$.

Notice that $u_{[\varphi\ell]_r}$ is a vector of length φ and $K_{r,s}\in GF(2)$, hence $T^{(s)}(u_{[\varphi\ell]})$ is a vector of length φ as required by the definition of the φth bit channel. The term $\oplus_{r=0}^{\ell-1}K_{r,s}\cdot u_{\varphi\ell+r}$ is simply the inner product of $u_{\varphi\ell}^{(\varphi+1)\ell-1}$ with the sth column of K.

From Lemma 1, it follows that a SC decoding algorithm at the kernel level, i.e., for a length-ℓ polar code with kernel K, that has time complexity t and space complexity s can be extended to a SC decoding algorithm for a length-n polar code with kernel K that has time complexity $O(tn\log n/(\ell\log\ell))$ and space complexity $O(sn/(\ell-1))$. In particular, there exists an implementation for the SC decoder that runs in $O(2^\ell n\log n/(\ell\log\ell))$. In practice, this time complexity may be too large even for relatively small values of ℓ. For this reason we propose to use a special type of kernel called a *permuted kernel* that can simultaneously reduce the time

[1]By the definition of polar codes, the values of the frozen bits are also required. If the channel is symmetric, then the frozen bits are all taken to be zero. For asymmetric channels, an assignment of the frozen bits that guarantees a vanishing probability of error is known to exist, however no practical method that finds such an assignment is known.

complexity of the SC decoder and achieve better scaling exponents.

A *permutation* ρ of length ℓ is a bijection $\rho : [\ell] \to [\ell]$. For a permutation ρ, the permutation matrix corresponding to ρ is denoted by M_ρ and defined by

$$(M_\rho)_{r,s} \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } s = \rho(r) \\ 0, & \text{otherwise,} \end{cases}$$

i.e., $u_0^{\ell-1} M_\rho = v_0^{\ell-1}$, where $v_{\rho(r)} = u_r$, for all $r \in [\ell]$. A *permuted kernel* with respect to ρ is defined by $K_\rho \stackrel{\text{def}}{=} M_\rho G_L$, where $G_L \stackrel{\text{def}}{=} G_{L,F_2}$ and $\ell = 2^L$. For ease of notation, for all $i \in [n]$ we denote the ith bit channel with respect to G_{m,K_ρ} by $\mathcal{W}_{m,\rho}^{(i)}$ and denote $\mathcal{W}_{m,F_2}^{(i)}$ by $\mathcal{W}_m^{(i)}$.

Let $\mathcal{C}_\rho$ be the polar code with kernel K_ρ and length $n = \ell$. For $i \in [\ell]$, let $D_i = \{\rho(0), \rho(1), \ldots, \rho(i-1), \rho(i)\}$ and let $d_i = \max\{D_i\}$.

Lemma 2. *For all $i \in [\ell]$, the ith bit channel with respect to K_ρ is equal to*

$$\mathcal{W}_{L,\rho}^{(i)}(y_0^{\ell-1}, u_0^{i-1}|u_i) = \sum_{v_{[d_i+1]\setminus D_i}} \mathcal{W}_L^{(d_i)}(y_0^{\ell-1}, v_0^{d_i-1}|v_{d_i}),$$

where $v_0^{\ell-1} = u_0^{\ell-1} M_\rho$, i.e., for all $r \in [\ell]$, $v_{\rho(r)} = u_r$.

Lemma 2 provides a connection between bit channels with respect to K_ρ and bit channels with respect to F_2, which will be useful for our SC decoding algorithm for $\mathcal{C}_\rho$, presented in Section III .

III. Formalization of SC Decoder for Permuted Kernels

In this section we formalize our SC decoder for a length-$\ell = 2^L$ code $\mathcal{C}_\rho$ defined by a permuted kernel K_ρ. As mentioned above, SC decoding for $\mathcal{C}_\rho$ is equivalent to SC decoding of a length-ℓ polar code with kernel F_2 that decides on the input bits in a permuted order according to ρ. For this reason, we call our SC decoding scheme *permuted successive cancellation* (PSC) decoding. We present an implementation of PSC decoding for any permutation ρ, which requires significantly less computational power and memory compared to the conventional SC decoder at the kernel level[2]. Analysis of time and space complexity is also presented.

The PSC decoding algorithm is similar to the list SC decoding from [13] in the sense that it computes many pairs of transition probabilities for some bit channels. Therefore, we will adapt some of the notation and terminology from [13]. In particular, for all $0 \le \lambda \le L$ define $\Lambda \stackrel{\text{def}}{=} 2^\lambda$. For $0 \le \varphi < \Lambda/2$ we have

$$\begin{aligned} \mathcal{W}_\lambda^{(2\varphi)}(z_0^{\Lambda-1}, b_0^{2\varphi-1}|b_\varphi) = \\ \sum_{b_{2\varphi+1}} \frac{1}{2} \mathcal{W}_{\lambda-1}^{(\varphi)}(z_0^{\Lambda/2-1}, b_{[2\varphi]_0} \oplus b_{[2\varphi]_1}|b_{2\varphi} \oplus b_{2\varphi+1}) \\ \cdot \mathcal{W}_{\lambda-1}^{(\varphi)}(z_{\Lambda/2}^{\Lambda-1}, b_{[2\varphi]_1}|b_{2\varphi+1}), \end{aligned} \tag{2}$$

[2]Our proposed algorithm admits better time complexity only when ρ is not the identity permutation. For the identity permutation the algorithm coincides with the conventional SC decoder.

and

$$\begin{aligned} \mathcal{W}_\lambda^{(2\varphi+1)}(z_0^{\Lambda-1}, b_0^{2\varphi}|b_{2\varphi+1}) = \\ \frac{1}{2} \mathcal{W}_{\lambda-1}^{(\varphi)}(z_0^{\Lambda/2-1}, b_{[2\varphi]_0} \oplus b_{[2\varphi]_1}|b_{2\varphi} \oplus b_{2\varphi+1}) \\ \cdot \mathcal{W}_{\lambda-1}^{(\varphi)}(z_{\Lambda/2}^{\Lambda-1}, b_{[2\varphi]_1}|b_{2\varphi+1}). \end{aligned} \tag{3}$$

Notice that, for $r \in \{0,1\}$, $[2\varphi]_r$ is the set of all elements in $[2\varphi]$ that are equal to r modulo-2. The index φ is called a *phase* and λ is called a *layer*. Thus, the pair of transition probabilities in phase i and layer λ is determined by two pairs of transition probabilities in phase $\varphi = \lfloor i/2 \rfloor$ and layer $\lambda - 1$; one corresponds to the output $(z_0^{\Lambda/2-1}, b_{[2\varphi]_0} \oplus b_{[2\varphi]_1})$ and the other to the output $(z_{\Lambda/2}^{\Lambda-1}, b_{[2\varphi]_1})$. From the recursive formulas above we obtain a binary tree of pairs of transition probabilities where the root of this tree is $\mathcal{W}_L^{(i)}(y_0^{\ell-1}, v_0^{i-1}|v_i)$, for some $i \in [n]$. Each pair of transition probabilities in this tree is associated with a *branch number* $0 \le \beta < 2^{L-\lambda}$. For $\lambda = L$ the branch number of $\mathcal{W}_\lambda^{(i)}(y_0^{\ell-1}, v_0^{i-1}|v_i)$ (the root of the tree) is 0. If the branch number of $\mathcal{W}_\lambda^{(i)}(z_0^{\Lambda-1}, b_0^i|b_i)$ is β then $\mathcal{W}_{\lambda-1}^{(\varphi)}(z_0^{\Lambda/2-1}, b_{[2\varphi]_0} \oplus b_{[2\varphi]_1}|b_{2\varphi} \oplus b_{2\varphi+1})$ and $\mathcal{W}_{\lambda-1}^{(\varphi)}(z_{\Lambda/2}^{\Lambda-1}, b_{[2\varphi]_1}|b_{2\varphi+1})$, $\varphi = \lfloor i/2 \rfloor$, have branch numbers 2β and $2\beta + 1$, respectively. With this terminology, we can refer to each pair of transition probabilities by a triple $(\varphi, \lambda, \beta)$ and denote

$$P_{\lambda,\beta}[b_0^\varphi] \stackrel{\text{def}}{=} \mathcal{W}_\lambda^{(\varphi)}(z_0^{\Lambda-1}, b_0^{\varphi-1}|b_\varphi).$$

We assign the same triple $(\varphi, \lambda, \beta)$ to the output and input of each pair of transition probabilities in the tree and denote $B_{\lambda,\beta}[\varphi] \stackrel{\text{def}}{=} b_\varphi$.

Remark 1. *We use the notations $(z_0^{\Lambda-1}, b_0^{\varphi-1})$ and b_φ for the output and input of the bit channel $W_\lambda^{(\varphi)}$ of branch number β, whereas the notations $(y_0^{\ell-1}, v_0^{\varphi-1})$ and v_φ are used only for $W_L^{(\varphi)}$. The values of $(z_0^{\Lambda-1}, b_0^{\varphi-1})$ and b_φ are determined recursively from $(y_0^{\ell-1}, v_0^{\varphi-1})$ and v_φ.*

For $i \in [\ell]$, let $\hat{u}_0^{i-1}$ be the bits decoded so far by the PSC decoding algorithm. Recall that in the ith step, the SC decoder calculates the pair of transition probabilities defined in (1). Denote these transition probabilities by

$$Q_i[b] \stackrel{\text{def}}{=} \mathcal{W}_{1,\rho}^{(i)}(y_0^{\ell-1}, \hat{u}_0^{i-1}|u_i = b).$$

Recall that $D_i = \{\rho(0), \rho(1), \ldots, \rho(i)\}$ and d_i is the maximum over D_i. The *decoding window* in the ith step is denoted by $DW_i = [d_i + 1] \setminus D_i$. By Lemma 2,

$$Q_i[b] = \sum_{v_s:\ s \in DW_i} P_{L,0}[v_0^i], \tag{4}$$

where $v_{\rho(i)} = u_i = b$ and for all $r \in [i]$, $v_{\rho(r)} = \hat{u}_r$. If $j \in DW_i$ then v_j was not determined yet and is therefore called an *unknown bit*. As with every SC decoder for polar codes, for every $i \in [\ell]$, the PSC decoding algorithm processes the ith step of the decoding by two stages:

1) Recursive transition probabilities computations.
2) Recursive decision making.

Next, we will describe these two stages.

1) Recursive transition probabilities computations:

In the beginning of the algorithm, where $i = 0$, $d_0 = \rho(0)$, and $DW_0 = [d_0 + 1]$, the algorithm recursively computes $P_{L,0}[v_0^{d_0}]$, for every choice of $v_0^{d_0-1}$. Thus, for every layer λ and any branch number β it calculates and stores the pair of transition probabilities that are required for the calculation of $P_{L,0}[v_0^{d_0}]$. Figure 1 illustrates the execution of this stage for $i = 0$ and a length-four permutation $\rho = (2, 0, 3, 1)$. For every $0 < i \leq \ell$, if $d_i = d_{i-1}$, then $P_{L,0}[v_0^{d_i}]$ was already computed, for every choice of v_{DW_i}, and the algorithm does not need to compute anything new. Otherwise, it must recursively calculate and store $P_{L,0}[v_0^{d_i}]$, for every choice of v_{DW_i}. The recursive calculation of $P_{\lambda,\beta}[b_0^j]$ is carried out through equations (2) and (3) (depending on the parity of j) using $P_{\lambda-1,2\beta}[b_{[2\varphi+1]_0} \oplus b_{[2\varphi+1]_1}]$ and $P_{\lambda-1,2\beta+1}[b_{[2\varphi+1]_1}]$, where $\varphi = \lfloor j/2 \rfloor$. Notice, that if the algorithm needs to calculate $P_{\lambda,\beta}[b_0^j]$ it will never make use of $P_{\lambda,\beta}[b_0^r]$, for $r < j$ and it can remove any such transition probability from the memory. Thus the algorithm stores $\sum_{\lambda=0}^{L} 2^{L-\lambda} = 2^{L+1} - 1 = 2\ell - 1$ vectors of transition probabilities pairs with various lengths.

2) Recursive decision making:

For every $0 \leq i < \ell$, after computing all the relevant pairs of transition probabilities in the previous stage, the algorithm computes $Q_i[b]$ according to (4) and decides on the value of $\hat{u}_i = v_{\rho(i)}$ according to the SC decoding decision rule, i.e. $\hat{u}_i = u_i$ if u_i is a frozen bit and otherwise it is set to the more likely value based on $Q_i[b]$, $b \in \{0, 1\}$. Once the value of $v_{\rho(i)}$ is determined, the algorithm recursively updates the values of the outputs/inputs for any other layers and branch numbers if these values are available. The recursive update of the inputs is carried out as follows. If $B_{\lambda,\beta}[j]$ was updated and j is odd then $B_{\lambda+1,2\beta+1}[\varphi] = B_{\lambda,\beta}[j]$, $\varphi = \lfloor j/2 \rfloor$, and the transition probabilities in layer $\lambda + 1$ and branch number $2\beta + 1$, associated with $B_{\lambda+1,2\beta+1}[\varphi] \neq B_{\lambda,\beta}[j]$ are removed. If both $B_{\lambda,\beta}[2\varphi]$ and $B_{\lambda,\beta}[2\varphi + 1]$ are available then $B_{\lambda+1,2\beta}[\varphi] = B_{\lambda,\beta}[2\varphi] \oplus B_{\lambda,\beta}[2\varphi+1]$ and the transition probabilities in layer $\lambda+1$ and branch number 2β associated with $B_{\lambda+1,2\beta}[\varphi] \neq B_{\lambda,\beta}[2\varphi] \oplus B_{\lambda,\beta}[2\varphi + 1]$ are removed.

An execution of this stage for $i = 0$ and the permutation $\rho = (2, 1, 3, 0)$ will result in only removing transition probabilities in layer 2. If, for example, the algorithm decision on the value of $v_{\rho(0)}$ was $v_{\rho(0)} = 0$, then the probabilities that are stored in layer 2 are $P_{2,0}[000]$, $P_{2,0}[010]$, $P_{2,0}[100]$, and $P_{2,0}[110]$. Repeating the recursive transition probabilities computation stage for $i = 1$ does not require any new transition probabilities computation since $d_1 = d_0 = 2$. Assuming the decision for $v_{\rho(1)}$ is 0 as well, at the decision-making stage the algorithm removes from layer 0 the transition probabilities associated with $v_{\rho(1)} = 1$. It then computes $B_{1,1}[0] = 0$ and removes the transition probabilities associated with $B_{1,1}[0] = 1$ from layer 1 and branch number 1. Since $\rho(1)$ is odd and $\rho(1) - 1$ was not yet calculated, it cannot make a decision for layer 1 and branch number 0. Figure 2 shows the result of propagating the decision $v_{\rho(1)} = 0$ to the layers and branch numbers that are affected by this decision.

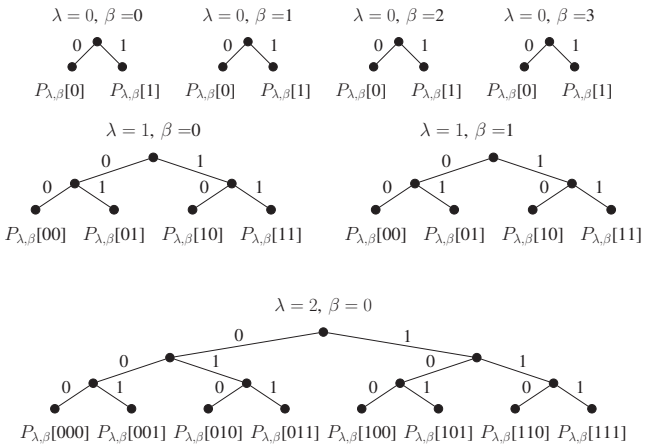

Fig. 1: Transition probabilities computations for each of the layers at step $i = 0$ and for $\rho = (2, 1, 3, 0)$.

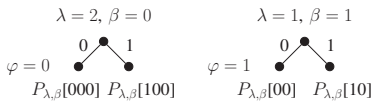

Fig. 2: The remaining transition probabilities in the affected layers and branch numbers at the end of step $i = 1$, after propagating the decisions $v_2 = 0$ and $v_1 = 0$, where $\rho = (2, 1, 3, 0)$.

IV. Simulation Results

In this section, we discuss the time and space complexity of our algorithm and compare the performance and decoding complexity of some polar codes with permuted kernels.

A. Time and Space Complexity

Both space complexity and time complexity of the algorithm are highly dependent on the permutation and therefore we express them as $O(s_\rho n)$ and $O(t_\rho n \log n)$, respectively, where s_ρ and t_ρ are constants that depend only on the permutation ρ. To compute t_ρ, we count the total number of pairs of transition probabilities that were calculated by the algorithm using equations (2) and (3), and divide this number by ℓ. Similarly, to compute s_ρ, we find the maximum number of transition probabilities pairs that were simultaneously stored in the memory and divide this number by ℓ.

Remark 2. *In the computation of the time complexity we ignore the computation of $Q_i[b]$, $i \in [\ell]$, by equation (4), given the transition probabilities pairs $P_{L,0}[v_0^i]$, since this requires at most $2t_\rho\ell$ operations. Similarly, for the computation of the space complexity we ignored the extra space used by the algorithm to store the phases of the unknown bits for each layer and branch number, as well as values of some known bits that are still being used. The space for this extra information is $o(s_\rho\ell)$.*

Unfortunately, there is no simple formula to compute s_ρ and t_ρ. Yet, we will show how to derive these quantities by considering a more complex example of a length-8 permutation $\rho = (1, 4, 0, 2, 7, 3, 6, 5)$. At step 0 the algorithm computes $P_{3,0}[v_0 v_1]$. To this end it needs to compute the 16 pair of transition probabilities $P_{\lambda,\beta}[b_0]$, for every $0 \leq \lambda < 3$ and $0 \leq \beta < 2^{3-\lambda}$. At step 1 it needs to compute $P_{3,0}[v_0^4]$ when v_1 is known, i.e., 2^3 new pairs of transition probabilities. To this end it must compute $P_{2,0}[b_0^2]$ and $P_{2,1}[b_0^2]$,

where $B_{2,1}[0] = v_1$. Thus, it must compute $4 + 2 = 6$ new pairs of transition probabilities at layer 2. At layer 1 it needs to compute $P_{1,\beta}[b_0^1]$, for every $0 \leq \beta < 4$, i.e., $2 \cdot 4 = 8$ new pairs. Overall it computes 22 pairs at this step. The algorithm will only compute new probabilities at step 4, where it computes 16 new pairs of transition probabilities. Overall it computes 54 pairs of transition probabilities and the decoding computation complexity of a length-n polar code with kernel K_ρ is $O(t_\rho n \log n)$, where $t_\rho = 2.25$. The maximum number of transition probabilities pairs that were stored in the memory at the same time is 30 and hence the space complexity of the algorithm in this example is $O(s_\rho n)$, where $s_\rho = 4.286$.

B. Simulation Results

Figure 3 depicts the performance comparison of three polar codes of length $n = 2^8$ over binary erasure channels. Two of these codes are constructed via K_σ, where

$$\sigma = (0, 1, 2, 3, 4, 6, 8, 10, 5, 9, 7, 11, 12, 13, 14, 15)$$

and K_π, where

$$\pi = (0, 1, 2, 4, 8, 3, 5, 6, 9, 10, 12, 7, 11, 13, 14, 15).$$

The other polar code is the conventional Arikan's polar code, constructed from the 2×2 kernel F_2. We chose these permuted kernels since they have relatively low scaling exponents, $\mu(K_\sigma) = 3.541$ and $\mu(K_\pi) = 3.479$. The time complexity of the PSC decoding algorithm for length-n polar codes with kernels K_σ and K_π is $O(t_\sigma n \log n)$ and $O(t_\pi n \log n)$, respectively, where $t_\rho = 1.907$ and $t_\pi = 2.407$. The space complexity for these codes is $O(s_\sigma n)$ and $O(s_\pi n)$, respectively, where $s_\sigma = 9.143$ and $s_\pi = 11.429$. It is observed that the polar code constructed from K_π and decoded with PSC outperforms both the conventional polar code and the code constructed by K_σ. The latter also outperforms the conventional polar code. The results are in agreement with the convention that the smaller the scaling exponent, the better the polar code performance. Notice that for some erasure channels the frame-error-rate of the polar code constructed from K_π is lower than the ML bound. Since the actual performance of the ML decoder can only be worse than the ML bound, the polar code constructed from K_π outperforms Arikan's polar code even when the latter is decoded by the ML decoder. It is to be noted that the conventional construction algorithms such as [14] cannot be applied directly for the kernels with $\ell \geq 4$ due to the exponential increase in the number of bit channel outputs. In this paper, we utilized a Monte-Carlo construction algorithm, which may be improved by using a larger number of iterations. We leave the study of low-complexity construction algorithms for future work.

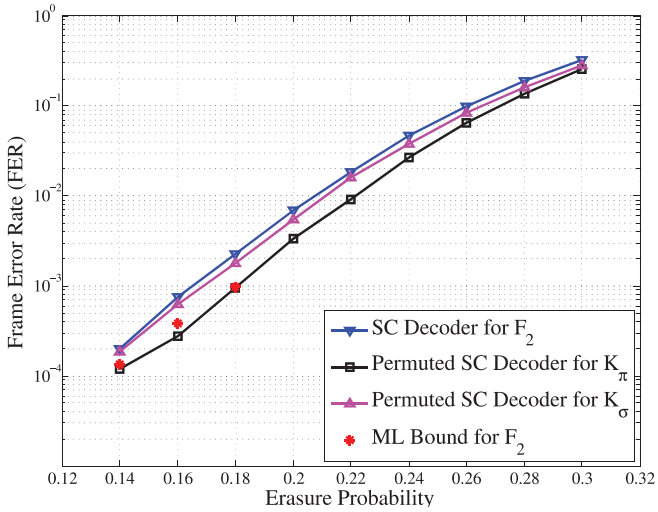

Fig. 3: FER comparison of three polar codes over binary erasure channels. The tree codes are constructed with F_2, K_σ, and K_π, respectively, at code length $n = 2^8$, and decoded with SC and PSC algorithms. All codes are optimized for the channel BEC(0.2) and rate $R = \frac{3}{5}$. The ML bound on performance of the F_2 polar code is also given, based upon methods in [13].

Acknowledgment

The work was supported in part by the National Science Foundation under grants CCF-1405119 and CCF-1619053 and by CMRR. The work of Sarit Buzaglo was supported in part by the ISEF Foundation and by the Weizmann Institute of Science - National Postdoctoral Award Program for Advancing Women in Science.

References

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. on Inform. Theory,* vol. 55, pp. 3051–3073, July 2009.

[2] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Commun. Letters*, vol. 12, pp. 447–449, June 2008.

[3] E. Arikan and E. Telatar, "On the rate of channel polarization," *Proc. IEEE Int. Symp. Inform. Theory,* pp. 1493–1495, Seoul, South Korea, July 2009.

[4] S. Buzaglo, A. Fazeli, P. H. Siegel, V. Taranalli, and A. Vardy, "On efficient decoding of polar codes with large kernels," to appear in *Proc. IEEE WCNC Workshop on Polar Coding*, San Fransisco, CA, March 2017.

[5] K. Chen, N. Kai, and L. Jiaru, "Improved successive cancellation decoding of polar codes," *IEEE Trans. on Commun.,* vol. 61, pp. 3100–3107, August 2013.

[6] U. U. Fayyaz and J. R. Barry, "Polar codes for partial response channels," *Proc. IEEE Int. Conf. Commun.,* Budapest, Hungary, pp. 4337–4341, June 2013.

[7] A. Fazeli and A. Vardy. "On the scaling exponent of binary polarization kernels, *Proc. Allerton Conf. on Commun., Control, and Computing,* October 2014.

[8] T. C. Gulcu, M. Ye, and A. Barg, "Construction of polar codes for arbitrary discrete memoryless channels," *Proc. IEEE Symp. Inform. Theory,* pp. 51–55, Barcelona, Spain, July 2016.

[9] J. Guo, M. Qin, A. Guillén i Fàbregas, and P. H. Siegel, "Enhanced belief propagation decoding of polar codes through concatenation," *Proc. IEEE Int. Symp. Inform. Theory,* pp. 2897–2991, Honolulu, Hawaii, July 2014.

[10] H. S. Hassani, K. Alishahi, and R. L. Urbanke, "Finite-length scaling of polar codes," *IEEE Trans. on Inform. Theory,* vol. 60, pp. 5875–5898, July 2014.

[11] Z. Huang, S. Zhang, F. Zhang, C. Duanmu, and M. Chen, "On the successive cancellation decoding of polar codes with arbitrary linear binary kernels," arXiv:1701.03264v2 [cs.IT], January 2017.

[12] S. B. Korada, E. Sasoglu, and R. L. Urbanke, "Polar codes: Characterization of exponent, bounds, and constructions," *IEEE Trans. on Inform. Theory,* vol. 56, pp. 6253–6264, December 2010.

[13] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. on Inform. Theory,* vol. 61, pp. 2213–2226, May 2015.

[14] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. on Inform. Theory,* vol. 59, pp. 6562–6582, October 2013.

[15] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Trans. on Commun.,* vol. 60, pp. 3221-3227, November 2012.

[16] M. Mondelli, R. L. Urbanke, and S. H. Hassani. "Unified scaling of polar codes: error exponent, scaling exponent, moderate deviations, and error floors," *IEEE Trans. on Inform. Theory,* vol. 62, pp. 6698–6712, December 2016.