Multi-Kernel Construction of Polar Codes

Frédéric Gabry, Valerio Bioglio, Ingmar Land, Jean-Claude Belfiore
Mathematical and Algorithmic Sciences Lab
France Research Center, Huawei Technologies Co. Ltd.
Email: {frederic.gabry,valerio.bioglio,ingmar.land, jean.claude.belfiore}@huawei.com

Abstract—We propose a generalized construction for binary polar codes based on mixing multiple kernels of different sizes in order to construct polar codes of block lengths that are not only powers of integers. This results in a multi-kernel polar code with very good performance while the encoding complexity remains low and the decoding follows the same general structure as for the original Arikan polar codes. The construction provides numerous practical advantages as more code lengths can be achieved without puncturing or shortening. We observe numerically that the error-rate performance of our construction outperforms state-of-the-art constructions using puncturing methods.

Index Terms—Polar Codes, Multiple Kernels, Successive Cancellation Decoding.

I. INTRODUCTION

Polar codes, recently introduced by Arikan in [1], are a new class of channel codes. They are provably capacity-achieving over various classes of channels, and they provide excellent error rate performance for practical code lengths. In their original construction, polar codes are based on the polarization effect of the Kronecker powers of the binary kernel matrix

\[ T_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}. \]

The generator matrix of a polar code is a sub-matrix of the transformation matrix \( T_2^\otimes n \).

In [1], Arikan conjectured that the polarization phenomenon is not restricted to the powers of the kernel \( T_2 \). This conjecture has been proven in [2], where necessary and sufficient conditions are presented for kernels \( T_l \) with \( l > 2 \) to allow for the polarization effect. This allowed researchers to propose polar code constructions based on larger kernels [3], both linear and non-linear [4]. Moreover, generalizations of binary kernels over larger alphabets can improve the asymptotic error probability [5]. Binary and non-binary kernels are mixed in [6], showing an additional improvement over homogeneous kernels construction. Equipped with all these techniques, it is possible to design codes based on the polarization effect of any size of the form \( N = l^n \) over various alphabets.

However, many block lengths cannot be expressed in the form \( N = l^n \). To overcome this limitation, puncturing [7] [8] and shortening [9] techniques have been proposed in the literature. Both shortening and puncturing techniques provide a practical way to construct polar codes of arbitrary lengths based on mother polar codes of length \( N = 2^n \), albeit with some disadvantages. First, punctured and shortened codes are decoded on the graph of their mother codes, and therefore the decoding complexity can be very high with respect to the shortened code length. Second, puncturing and shortening may lead to a substantial loss in terms of polarization speed, and hence a worse error-rate performance. Finally, the lack of structure of the frozen sets and puncturing or shortening patterns generated by these methods makes them non-suitable for practical implementation.

In this paper, we propose a generalized construction of polar codes based on mixing of kernels of different sizes over the same binary alphabet, in order to construct polar codes of any block length. By using kernels of different sizes in different stages, it is in fact possible to construct polar codes of block lengths that are not only powers of integers. This is a major difference over other mixed constructions, e.g. [6], designed to optimize the performance of polar codes of block length restricted to powers of integers. With our construction, we obtain a new family of polar codes, coined multi-kernel polar codes in the following, with error-correcting performance comparable or even better than state-of-the-art polar codes obtained via puncturing/shortening methods. Furthermore, the encoding complexity is similar to polar codes while the decoding follows the same general structure. Therefore, list decoding algorithms can be used for multi-kernel polar codes, which can as well be enhanced by the use of cyclic redundancy check (CRC) bits [10]. In the following, we show an example of multi-kernel polar construction mixing kernels of sizes 2 and 3, which allows for code lengths \( N = 2^{n_2} \cdot 3^{n_3} \) without puncturing or shortening.

This paper is organized as follows. In Section II we present our general construction, including the encoding and decoding, of multi-kernel polar codes. In Section III we describe explicitly the construction for the case of mixed binary kernels of size 2 and 3. In Section IV we illustrate numerically the performance of the codes, and Section V concludes this paper.

II. MULTI-KERNEL CONSTRUCTION

In this section, we describe the proposed multi-kernel polar codes construction for a code of length \( N = n_1 \cdots n_s \), with \( n_i \) not being necessarily distinct prime numbers. Each integer \( n_i \) corresponds to a binary kernel \( T_{n_i} \), of size \( n_i \), i.e., a squared \( n_i \times n_i \) binary matrix [3]. The transformation matrix of such a multi-kernel polar code is defined as \( G_N = T_{n_1} \otimes \cdots \otimes T_{n_s} \).

Note that a different ordering of the kernels would drive to a different transformation matrix, as the Kronecker product is not commutative.

A. Tanner Graph Construction

The Tanner graph of multi-kernel polar codes is a generalization of the Tanner graph of polar codes, shown in Figure II.
Finally, the edges between Stage 2 and Stage 3, we recall that 1 of stage 2, bit 2 is connected to bit 3, etc. To construct, we calculate the canonical permutation \( Q \). On the other hand, for multi-kernel polar codes, \( Q \) is the inverse of the product of the other permutations, i.e., \( P_1 = (P_2 \cdot \ldots \cdot P_s)^{-1} \). The other permutations are based on a basic permutation, that we call \( Q_1 \)-canonical permutation. If \( N_i = \prod_{j=1}^{s-1} n_j \) is the partial product of the kernel sizes at stage \( i \), then \( Q_i \) is a permutation of \( N_i \times 1 \) elements defined as in Equation (1) at the top of next page. Finally, \( P_s = (Q_1 \cdot Q_2 + N_1 + 1 \cdot Q_2 + N_2 + N_2 + 1 \cdot \ldots \cdot Q_s + (N/s+1) \cdot 1) \). Note that for the last stage \( P_s = Q_s \). An example of this construction is shown in Figure 2.

In order to clarify the description of the graph construction, let us construct the graph in Figure 1 i.e., for the \( G_8 = T_2 \otimes T_2 \otimes T_2 \) transformation matrix, using the proposed algorithm. We first draw 3 stages, each constituted of \( N/2 = 4 \) T2 boxes. To construct the edges between Stage 1 and Stage 2, we calculate the canonical permutation \( Q_2 = 1 2 3 4 \ 1 3 2 4 \). Given the canonical permutation, we can calculate \( P_2 = (Q_2 \cdot N_3 + 1) = 2 3 4 5 6 7 8 \). We observe indeed, in Figure 1 that bit 1 of stage 1 is connected to bit 1 of stage 2, bit 2 is connected to bit 3, etc. To construct the edges between Stage 2 and Stage 3, we recall that \( P_3 = Q_3 \), and hence \( P_3 = 1 2 3 4 5 6 7 8 \). Finally, \( P_1 = (P_2 \cdot P_3)^{-1} = 1 2 3 4 5 6 7 8 \).

### B. Encoding of Multi-Kernel Polar Codes

As for polar codes, a multi-kernel polar code of length \( N \) and dimension \( K \) is completely defined by a transformation matrix \( G_N = T_{n_1} \otimes \ldots \otimes T_{n_s} \) and a frozen set \( F \). We recall that, in contrast to polar codes, the order of the factors of the Kronecker product is important, since different orderings result in different transformation matrices, with different polarization behaviors and hence different frozen sets. Given a kernel order, the reliabilities of the bits can be calculated for a target SNR through a Monte-Carlo method or the density evolution algorithm II on the resulting transformation matrix. We select the order of the kernels summing the reliabilities of the \( K \) best bits of each ordering, keeping the kernel order resulting in the largest sum. When the ordering of the kernels is decided, along with the corresponding transformation matrix of the code, the frozen set \( F \) is given by the \( N-K \) less reliable bits. To simplify the notation, in the following we assume the kernels in the Kronecker product to be already ordered.

Hereafter, the \( K \) information bits are stored in the length- \( N \) message \( u \) according to the frozen set, i.e., they are stored in the positions not belonging to \( F \), while the remaining bits are filled with zeros. Finally, a codeword \( x \) of length \( N \) is obtained as \( x = uG_N \). We notice that the encoding may be performed on the Tanner graph of the multi-kernel polar code. In Section III we will give an example of this construction for the case \( N = 6 \).

### C. Decoding of Multi-Kernel Polar Codes

The decoding of multi-kernel polar codes is performed through successive cancellation (SC) decoding on the Tanner graph of the code, similarly to polar codes. In general, the log-likelihood ratios (LLRs) are passed along the Tanner graph from the right to the left, while the hard decisions on the decoded bits are passed from the left to the right. Assuming that \( G_N = T_{n_1} \otimes \ldots \otimes T_{n_s} \), the LLRs go through \( B_1 \) \( T_{n_1} \)-boxes of size \( n_1 \times n_1 \) on the first stage, until the \( B_s \) \( T_{n_s} \)-boxes of size \( n_s \)-by-\( n_s \) on the last stage. We call LLR \( (j, i) \) and \( u(j, i) \) the values taken by the LLR and the hard decision of bit \( i \) at stage \( j \) of the Tanner graph respectively. The update of the LLRs is then done according to update functions corresponding to the kernel used at a given stage. The description of this section will be clarified in Section III with a detailed description of the update functions for LLRs and hard decisions for kernels \( T_2 \) and \( T_3 \).

SC decoding operates as follows. Initially, the LLRs of the coded bits \( x_i \) based on the received vector \( y \) are calculated at the receiver. The received signal is decoded bit-by-bit using LLR propagation through the graph to retrieve the transmitted message \( u = [u_0, \ldots, u_i, \ldots, u_{N-1}] \). For every bit \( u_i \), the nature of its position \( i \) is initially checked. If \( u_i \) is a frozen bit, it is decoded as \( \tilde{u}_i = 0 \), and the decoder moves on to the next bit. If \( u_i \) is an information bit, its LLR is recursively calculated to make a hard decision, starting by LLR \( (s, i) \). In general, the calculation of LLR \( (j, l) \) is done using the LLR \( (j-1, \cdot) \) and the hard decisions \( u(j, \cdot) \) that are connected to the \( T_{n_i} \) box outputting LLR \( (j, l) \). The value of LLR \( (j, l) \) is calculated.
Q_i = \begin{pmatrix} 1 & 2 & \ldots & N_i \\ n_i + 1 & (N_i - 1)n_i + 1 & \ldots & N_i + 1 \\ \end{pmatrix}. \quad (1)

According to the update rules corresponding to the \( T_{n_2} \) kernel. These update rules are given in Section III for the case of kernels \( T_2 \) and \( T_3 \) used in the numerical illustration of the code construction. Using such a recursive procedure, the algorithm will arrive to the calculations of \( \text{LLR}(1, \cdot) \) values in the graph, which are obtained using the channel LLRs and the updating rules corresponding to \( T_{n_2} \). Hence all the LLRs needed for the computation of \( \text{LLR}(s, i) \) are eventually calculated, and \( \text{LLR}(s, i) \) is computed. Finally, the bit is decoded as \( \tilde{u}_i \) given by the hard decision corresponding to the sign of \( \text{LLR}(s, i) \).

### III. Example: Mixing \( T_2 \) and \( T_3 \) Kernels

In this section we illustrate the general code construction through a simple example using kernels \( T_2 \) and \( T_3 \), which allow us to construct codes for any length expressed as \( N = 2^{a_2} \cdot 3^{a_3} \). In particular, we will illustrate the encoding and decoding of the proposed codes using a \( N = 6 \) code with transformation matrix \( G_6 = T_2 \otimes T_3 \), see Figure 2. We recall that the code construction is more general, and multi-kernel polar codes mixing binary kernels of any size can be designed following the construction described in Section II. We initially describe the updating rules for the kernels used in our implementation. Even if the update rules for \( T_2 \) are well-known as the canonical component of original polar codes, the update rules for \( T_3 \) kernel are mostly unknown in the literature. Then, we show how to construct the Tanner graph of multi-kernel polar codes based on binary kernels of sizes 2 and 3. Finally, we show how the SC decoding procedure described previously is applied to the Tanner graph of the code.

#### A. Decoding rules and functions for \( T_2 \) kernels

In this section we remind the well-known decoding rules for the Arikan polar codes’ kernel [1]. The fundamental \( T_2 \) block can be depicted as

\[
\begin{array}{c}
\begin{pmatrix}
0 \\
1
\end{pmatrix} & \begin{pmatrix}
1 \\
0
\end{pmatrix} & \begin{pmatrix}
0 \\
1
\end{pmatrix} & \begin{pmatrix}
1 \\
0
\end{pmatrix}
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3
\end{pmatrix}
= \begin{pmatrix}
0, \lambda_0 \\
1, \lambda_1 \\
0, \lambda_2 \\
1, \lambda_3
\end{pmatrix}
\begin{pmatrix}
x_0, l_0 \\
x_1, l_1 \\
x_2, l_2 \\
x_3, l_3
\end{pmatrix}
\]

where \( (u_0, u_1) \cdot T_2 = (x_0, x_1) \), with \( T_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \), and where \( \lambda_i \) and \( l_i \) denote the LLRs and \( u_i \) and \( x_i \) denote the hard decisions on the bits. This corresponds to the hard-decision update rules

\[
x_0 = u_0 \oplus u_1, \\
x_1 = u_1 \oplus x_1.
\]

The inverse of the update rules are \( u_0 = x_0 \oplus x_1 \) and \( u_1 = x_1 = u_0 \oplus x_0 \), corresponding to the message update equations

\[
\lambda_0 = l_0 \oplus l_1, \\
\lambda_1 = (-1)^{u_0} \cdot l_0 + l_1,
\]

where \( a \oplus b \triangleq 2 \tanh^{-1}(\tanh(a/2) \cdot \tanh(b/2)) \simeq \text{sign}(a) \cdot \text{sign}(b) \cdot \min(|a|, |b|) \).

#### B. Decoding rules and functions for \( T_3 \) kernels

In this section we describe the decoding rules for the \( T_3 \) kernel used in our design. A study of the design of kernels of size 3 can be found in [12]. The fundamental \( T_3 \) block can be depicted as follows

\[
\begin{array}{c}
\begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix} & \begin{pmatrix}
1 \\
0 \\
1
\end{pmatrix} & \begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix} & \begin{pmatrix}
1 \\
0 \\
1
\end{pmatrix}
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3
\end{pmatrix}
= \begin{pmatrix}
0, \lambda_0 \\
1, \lambda_1 \\
0, \lambda_2 \\
1, \lambda_3
\end{pmatrix}
\begin{pmatrix}
x_0, l_0 \\
x_1, l_1 \\
x_2, l_2 \\
x_3, l_3
\end{pmatrix}
\]

where \( (u_0, u_1, u_2) \cdot T_3 = (x_0, x_1, x_2) \), with \( T_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \).

Consequently, the hard-decisions update rules are

\[
x_0 = u_0 \oplus u_1, \\
x_1 = u_0 \oplus u_2, \\
x_2 = u_0 \oplus u_1 \oplus u_2.
\]

Inverting these equations, we get \( u_0 = x_0 \oplus x_1 \oplus x_2, u_1 = u_0 \oplus x_0 = x_1 \oplus x_2 \) and \( u_2 = u_0 \oplus x_1 = u_0 \oplus u_1 \oplus x_2, \) from which we get the message update equations

\[
\lambda_0 = l_0 \oplus l_1 \oplus l_2, \\
\lambda_1 = (-1)^{u_0} \cdot l_0 + l_1 \oplus l_2, \\
\lambda_2 = (-1)^{u_0} \cdot l_1 + (-1)^{u_0 \oplus u_1} \cdot l_2.
\]

#### C. Construction Example for \( G_6 = T_2 \otimes T_3 \)

The Tanner graph of the length-6 code obtained from the \( G_6 = T_2 \otimes T_3 \) transformation matrix is shown in Figure 2. Let \( u = [u_0, \ldots, u_5] \) be the bits to be encoded, \( K \) of which are information bits and \( 6 - K \) are frozen bits (according to the frozen set \( F \)). The codeword \( x = [x_0, \ldots, x_5] \) is then

Fig. 2. Tanner graph of the multi-kernel polar code with \( N = 6 \) and \( G_6 = T_2 \otimes T_3 \).
constructed as $x = u \cdot G_0$, or following the hard-decision update rules of the Tanner graph.

The decoding starts with $u_0$. If $u_0$ is a frozen bit, its value $\hat{u}_0$ is set to 0 and the decoding continues with $u_1$. Otherwise, $\text{LLR}(2, 0)$ is calculated to make a hard decision on the value of $u_0$. According to the Tanner graph and the decoding rules of the $T_3$ kernel, $\text{LLR}(2, 0) = \text{LLR}(1, 0) \oplus \text{LLR}(1, 2) \oplus \text{LLR}(1, 4)$. The values of these intermediate LLRs have to be calculated. According to the update rules of kernel $T_2$ and the Tanner graph of the multi-kernel polar code, $\text{LLR}(1, 0) = \text{LLR}(0, 0) \oplus \text{LLR}(0, 3)$, while $\text{LLR}(1, 2) = \text{LLR}(0, 1) \oplus \text{LLR}(0, 4)$ and $\text{LLR}(1, 4) = \text{LLR}(0, 2) \oplus \text{LLR}(0, 5)$. Since LLRs at stage 0 are the LLRs of the received signal, the recursion stops and $\text{LLR}(2, 0)$ is calculated. We notice that, in order to speed up the decoding, the values of the LLRs and hard-decisions at the intermediate stages can be stored, since they remain constant during the decoding once calculated.

IV. Numerical Illustrations

In the following, we show the performance of multi-kernel polar codes for the kernels described in Section III, i.e., where $T_2$ and $T_3$ kernels are mixed.

In particular, we show the performance of the multi-kernel polar codes of length $N = 72 = 2^3 \cdot 3^2$ and $N = 48 = 2^4 \cdot 3$. For $N = 72$ there exist 10 possible permutations of the kernels $T_2$ and $T_3$, i.e., 10 different ways to construct the transformation matrix of the multi-kernel polar code. For $K = 36$ information bits, density evolution analysis suggests to use the transformation matrix $G_{72} = T_3 \otimes T_2 \otimes T_2 \otimes T_2$ to build the code. For $N = 48$, only 5 permutations of the kernels $T_2$ and $T_3$ can be generated, and the transformation matrix $G_{48} = T_2 \otimes T_2 \otimes T_2 \otimes T_2$ has been selected by the density evolution analysis for $K = 24$. Both codes are designed for a rate equal to 1/2 and SNR of 2 dB.

In Figure 3 and in Figure 4 we compare the BLock Error Rate (BLER) performance of the proposed multi-kernel polar code for an additive white Gaussian noise (AWGN) channel against state-of-the-art punctured and shortened polar codes, proposed in [7] and [9], respectively. For $N = 72$ a mother polar code of length $N' = 128$ is used, while for $N = 48$ the mother polar code has length $N' = 64$, both designed for the same target SNR of 2 dB. In the figures, we show the SC-List decoding performance of the codes for list size $L = 8$ and $L = 1$, the latter corresponding to SC decoding.

We observe that for both cases, our construction significantly outperforms state-of-the-art punctured and shortened polar codes. Moreover, multi-kernel polar codes exhibit a smaller decoding complexity compared to punctured/shortened polar codes, due to their reduced code length construction. In fact, SC decoding of a punctured/shortened polar code has to be performed over the Tanner graph of its mother polar code, i.e., the complexity of the code depends on the length of the mother code. An estimation of the complexity is calculated as the number of LLRs calculated during the decoding, given by the block length multiplied by the number of stages of the graph. For multi-kernel polar codes, $N \cdot s$ LLRs are calculated, while for punctured/shortened polar codes $N' \log_2 N'$ LLRs are calculated, with $N' = 2^\lceil \log_2 N' \rceil$ since the decoding is performed over the graph of the mother code. In particular, for length-72 punctured/shortened polar codes, the calculation of 896 LLRS is required, compared to only 360 LLRs for multi-kernel polar codes, which shows a substantial complexity reduction.

V. Conclusions

In this paper, we proposed a generalized polar code construction based on the polarization of multiple kernels. The construction provides numerous practical advantages, as more code lengths can be achieved without puncturing or shortening and only modest puncturing and shortening is required to achieve any arbitrary code length. For an example with binary kernels of size 2 and 3, we observed numerically that the error-rate performance of our construction clearly outperforms state-of-the-art constructions using puncturing or shortening methods.
REFERENCES

[1] E. Arikan, “Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[2] S. B. Korada, E. Sasoglu, and R. Urbanke, “Polar codes: Characterization of exponent, bounds, and constructions.” *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6253–6264, Dec. 2010.

[3] N. Presman, O. Shapira, S. Litsyn, T. Etzion, and A. Vardy, “Binary polarization kernels from code decompositions,” *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2227–2239, May 2015.

[4] H.-P. Lin, S. Lin, and K. Abdel-Ghaffar, “Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents,” *IEEE Transactions on Information Theory*, vol. 61, no. 10, pp. 5253–5270, Oct. 2015.

[5] R. Mori and T. Tanaka, “Non-binary polar codes using Reed-Solomon codes and algebraic geometry codes,” in *IEEE Information Theory Workshop*, Dublin, Ireland, Sep. 2010.

[6] N. Presman, O. Shapira, and S. Litsyn, “Mixed-kernels constructions of polar codes,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 239–253, Feb. 2016.

[7] K. Niu, K. Chen, and J.-R. Lin, “Beyond turbo codes: Rate-compatible punctured polar codes,” in *IEEE International Conference on Communications (ICC)*, Budapest, Hungary, June 2013.

[8] L. Zhang, Z. Zhang, X. Wang, Q. Yu, and Y. Chen, “On the puncturing patterns for punctured polar codes,” in *IEEE International Symposium on Information Theory (ISIT)*, Hawaii, U.S.A., July 2014.

[9] R. Wang and R. Liu, “A novel puncturing scheme for polar codes,” *IEEE Communications Letters*, vol. 18, no. 12, pp. 2081–2084, Dec. 2014.

[10] I. Tal and A. Vardy, “List decoding of polar codes,” in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, St. Petersburg, Russia, July 2011.

[11] R. Mori and T. Tanaka, “Performance of polar codes with the construction using density evolution,” *IEEE Communications Letters*, vol. 13, no. 7, pp. 519–521, July 2009.

[12] L. Zhang, Z. Zhang, and X. Wang, “Polar code with block-length $N = 3^n$,” in *IEEE International Conference on Wireless Communications & Signal Processing (WCSP)*, Huangshan, China, Oct. 2012.