Software Model: Multi-Kernel Polar Code Decoders

Mushfiqur Rahman
Department of Electrical and Computer Engineering
McMaster University, Hamilton, Ontario, Canada
Email: rahmam8@mcmaster.ca

***Abstract*—Polar codes are a class of error correcting codes that achieve channel capacity as their block lengths approach infinity. The low complexity of the encoder, decoder and the explicit construction of polar codes have paved the way for their adoption into the 5G NR standard by 3GPP. This work aims to summarize the fundamentals of Arikan polar codes, and multi-kernel polar code. The operation of the Successive-Cancellation Decoder is summarized, as well as that of the Fast-SSC decoder. Two code construction methods for multi-kernel polar codes are also reviewed: the Gaussian Approximation method and the Minimum Distance Construction. Lastly, the operation and results of a software model to simulate these code constructors, encoders, and decoders are presented.**

I. Introduction

Polar codes are the first family of error correcting codes that provably achieve channel capacity through the concept of channel polarization [1]. To provide an example for channel polarization, consider a channel *W*. Channel polarization is a technique to produce N new copies of *W* where some channels of the N channels are extremely noisy (W^-) while others are noiseless (W^+). These N channels are called bit-channels. For a finite block length N and rate R = K/N, a polar code can be constructed by choosing K least noisy channels out of the N channels. The indices of the K least noisy channels are called the Information Set ($\mathcal{I}$), and the remaining N – K indices are called the Frozen Set ($\mathcal{F}$). The block length (N), message length (K) and the frozen set, define a polar code. As $N \to \infty$, the fraction of reliable bits of the N bits approaches the channel capacity of *W*. The polarization itself is done through Kronecker powers of the 2×2 polarization kernel matrix: $\boldsymbol{T_2} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$.

Polar codes have an $O(\mathrm{N}\log_2\mathrm{N})$ complexity for encoding and decoding, when the decoding is done using the original successive cancellation (SC) algorithm as proposed in [1]. As a result, Polar Codes have been selected for use in the 5G standard as the error control code for the control channel [2]. It should be noted however that the 5G standard focuses on short block length polar codes, with a maximum block length of 1024 bits.

There are three major drawbacks to polar codes:

- Low throughput due to the serial nature of the successive decoding algorithm
- Inferior error correction performance when compared to LDPC and Turbo codes of the same length
- Polar codes can only be constructed with block lengths as powers of two.

Numerous algorithms have been proposed to tackle the low throughput issue. In [3] a Simplified Successive Cancellation algorithm is proposed, which identifies two linear sub codes: a Rate-0 and a Rate-1 code. Rate-1 sub codes are codes where the index of the sub-code bit within the overall code is part of the information set, while Rate-0 sub-codes are those where the index of the sub-code bit within the overall code is part of the information set. During the decoding process, these sub-codes can be decoded significantly faster. In [4] and [5], more linear sub codes are identified to improve throughput.

In [6] a new decoder was proposed, named Successive-Cancellation List (SCL) Decoder. List decoding showed a considerable improvement in error correction performance especially when the polar code is concatenated with a CRC. The SCL decoder has complexity $O(\mathrm{LN}\log_2\mathrm{N})$ where L is the list size.

The length constraint primarily arises from the fact that, any codes produced by any Kronecker power of the T_2 matrix will be a power of two. Arikan conjectured in [1] that the polarization phenomenon is not restricted to powers of the T_2 kernel matrix, and this was proven in [7]. As a result, codes could now be constructed with block lengths equal to powers of prime numbers. Since codes cannot always be constructed with block lengths being powers of primes, rate puncturing and shortening techniques were introduced in [8], [9] and [10] to construct codes of any length at the cost of worse error correction performance. Additionally, punctured and shortened codes have a very high decoding complexity. In [14] a method of polar code encoding, and decoding was proposed which used multiple kernels of different sizes to increase the number of code lengths that can be constructed without puncturing or shortening. For example, a $N = 2^1 \cdot 3^2 = 18$ code can be constructed using a generator matrix: $G_1 = T_2 \otimes T_3 \otimes T_3$ matrix where $\otimes$ matrix denotes the Kronecker product, and T_3 is the 3×3 polarization kernel given by: $\boldsymbol{T_3} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. Note that since the Kronecker product is not commutative, the generator matrix can also be $G_2 = T_3 \otimes T_3 \otimes T_2$, or $G_3 = T_3 \otimes T_2 \otimes T_3$. Codewords produced by using generator matrix G_1 are different from those produced by G_2, which are different from those by G_3. As a result, multi-kernel polar codes are defined by N, K, the frozen set, and the ordering of the kernels.

A. Min-Sum Approximation

A Single Parity Check code is a (N, N-1) linear block code. Given a message vector $\boldsymbol{m} = [m_1\ m_2\ \ldots m_{n-1}]$, the codeword is given as $\boldsymbol{c} = [m_1\ m_2\ \ldots m_{n-1}\ p]$, where p is the parity calculated as: $p = m_1 \oplus m_2 \oplus \ldots \oplus m_{n-1}$.

Consider a message vector of length two: $\boldsymbol{m} = [m_1\ m_2]$. This message vector is encoded to form the codeword $\boldsymbol{c} = [c_1\ c_2\ c_3]$, which is then BPSK modulated, and transmitted over an AWGN channel. On the receiving end, the received vector is $\boldsymbol{r} = [r_1\ r_2\ r_3]$, with $r_1, r_2, r_3 \in \mathbb{R}$. The objective now is to decode this received vector using a soft decoder.

We would like to use the received values of r_1 and r_2 to make a guess on the value of c_1. Let's denote the probability that c_2 is zero, given the received value of r_2 to be:

$\mathrm{P}(c_2 = 0|r_2) = P_2$ and $P(c_3 = 0|r_3) = P_3$. Therefore, the beliefs for c_2 and c_3 are:

$$l_2 = \ln\left[\frac{P(c_2 = 0\,|r_2)}{P(c_2 = 1\,|r_2)}\right] = \ln\left(\frac{P_2}{1-P_2}\right) \propto r_2$$

$$l_3 = \ln\left[\frac{P(c_3 = 0\,|r_3)}{P(c_3 = 1\,|r_3)}\right] = \ln\left(\frac{P_3}{1-P_3}\right) \propto r_3$$

Additionally, we know that $c_1 = c_2 \oplus c_3$. Constructing a truth table for c_1, we get:

c_2	c_3	c_1
0	0	0
0	1	1
1	0	1
1	1	0

As a result, we get the following conditional probabilities for c_1:

$$P_1 = \mathrm{P}(c_1 = 0|r_2, r_3) = P_2P_3 + (1-P_2)(1-P_3)$$

$$(1-P_1) = \mathrm{P}(c_1 = 1|r_2, r_3) = P_2(1-P_3) + (1-P_2)P_3$$

Using these expressions, we can calculate the log-likelihood ratio (LLR) of c_1, denoted by L_1, to be:

$$L_1 = \ln\left(\frac{P_1}{1-P_1}\right) = \ln\left[\frac{\mathrm{P}(c_1 = 0|r_2, r_3)}{\mathrm{P}(c_1 = 1|r_2, r_3)}\right]$$

Simplifying:

$$\frac{P_1}{1-P_1} = \frac{P_2P_3 + (1-P_2)(1-P_3)}{P_2(1-P_3) + (1-P_2)P_3}$$

$$\frac{1-\left(\frac{1-P_1}{P_1}\right)}{1+\left(\frac{1-P_1}{P_1}\right)} = \frac{1-\left(\frac{1-P_2}{P_2}\right)}{1+\left(\frac{1-P_2}{P_2}\right)} \cdot \frac{1-\left(\frac{1-P_3}{P_3}\right)}{1+\left(\frac{1-P_3}{P_3}\right)}$$

$$\frac{1-e^{-L_1}}{1+e^{-L_1}} = \frac{1-e^{-l_2}}{1+e^{-l_2}} \cdot \frac{1-e^{-l_3}}{1+e^{-l_3}}$$

$$\tanh\left(\frac{L_1}{2}\right) = \tanh\left(\frac{l_2}{2}\right) \cdot \tanh\left(\frac{l_3}{3}\right)$$

$$\therefore L_1 = 2\,\mathrm{arctanh}\left(tanh\left(\frac{l_2}{2}\right) \cdot tanh\left(\frac{l_3}{2}\right)\right) \rightarrow (1)$$

Through these calculations, we have essentially shown how to calculate the belief of c_1 given the received values of r_2 and r_3, when $c_1 = c_2 \oplus c_3$. In a similar manner, we can derive L_2 and L_3. The issue now is to find a good approximation for Eqn. (1). In [13] it was shown that a good approximation is: $sign(l_2) \cdot sign(l_3) \cdot \min(|l_2|, |l_3|)$. This is known as min-sum approximation and will be used for the decoding of polar codes.

II. Polar Codes

Polar codes, denoted by $P(N, K, \mathcal{F})$ are linear block codes with a block length N, message length K, a frozen set $(\mathcal{F})$ and an information set $(\mathcal{I})$. The indices of the K most reliable bit-channels after channel polarization comprise the information set, and the remaining N – K indices comprise the frozen set.

Polar Code construction is the process of finding the K most reliable indices to place the message bits in. A comparison of four current code constructions methods is presented in [11].

Once the frozen set is determined, the source-word vector $\boldsymbol{u}$ is formed by placing the K message bits into indices corresponding to the information set, while the remaining indices corresponding to the frozen set are set to 0. The source-word can then be encoded by computing: $\boldsymbol{x} = \boldsymbol{u} \cdot \boldsymbol{G}$ where $\boldsymbol{G}$ is the generator matrix given by $\boldsymbol{G} = \boldsymbol{T}_2^{\otimes n}$, and $A^{\otimes n}$ denotes the Kronecker product of A with itself, taken n times. Also, $A^{\otimes 0} = 1$. All calculations are done over GF(2) and so the encoding process can be represented by a network of XOR operations on the source-word vector. The $\boldsymbol{T}_2$ is the 2×2 polarizing kernel as proposed by Arikan and as a result will be referred to as the Arikan kernel.

For a source-word with length 2, we can find the codeword to be:

$$\begin{aligned}\boldsymbol{x} &= \boldsymbol{u} \cdot \boldsymbol{G} \\ &= \boldsymbol{u} \cdot \boldsymbol{G} \\ &= [u_0\ u_1] \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \\ &= [u_0 \oplus u_1\ \ u_1]\end{aligned}$$

$$\begin{aligned}x_0 &= u_0 \oplus u_1 \\ x_1 &= u_1\end{aligned}$$

This operation can is illustrated in Fig. 1, with $\oplus$ representing an XOR operation.

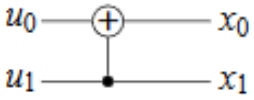

Fig. 1:XOR operations for encoding polar codes of length 2 (image based on Fig. 1 in [4])

The 2×2 block in Fig. 1 can be stacked horizontally to form bigger codewords of length $2^n, n \in \mathbb{N}$. This is shown in Fig. 2 for an N = 8 polar code. The number of stages can be calculated as $\log_2 N$ when the code is comprised solely of Arikan kernels.

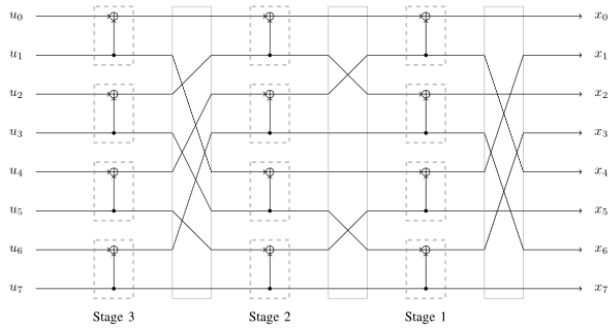

Fig. 2: Stacking of Arikan Kernels (image taken from [12])

In between each stage permutations are inserted, which are simply the bit indices of the input to the current stage cyclically right shifted by one [1]. For example, for the permutation between Stage 3 and Stage 2, index 1 corresponds to 001 in binary, which cyclically right shifted results in 100, corresponding to bit index 4.

A. *Successive-Cancellation (SC) Decoding*

Successive-Cancellation algorithm for decoding polar codes is described in [1]. The decoding process can be

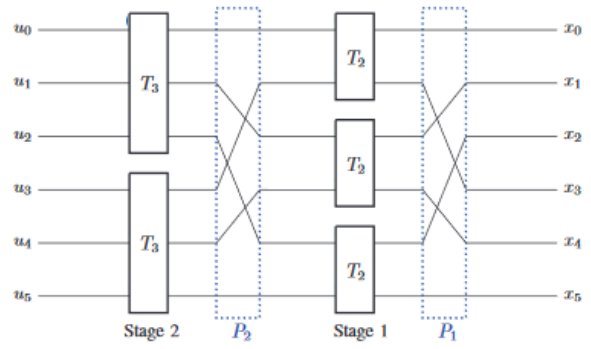

Fig. 4: Tanner Graph of a multi-kernel polar code with N = 6 and $G_6 = T_2 \otimes T_3$ (image based on [14])

visualized as a tree traversal with a priority placed on traversing left nodes first. A decoding tree for a P(8, 4, {0, 1, 2, 4}) polar code is shown in Fig. 3(a). The stage number is represented by s, $\hat{\boldsymbol{u}}$ is the decoded vector, the frozen bits are grey, and the information bits are white. It can be observed that a polar code of length N is essentially the concatenation of two polar codes of length N/2.

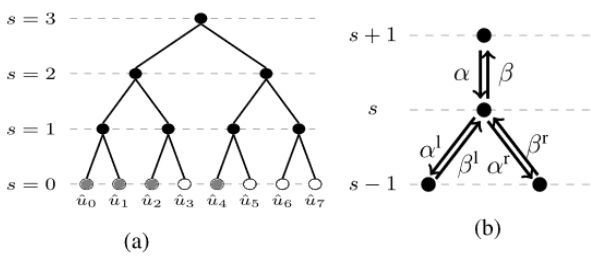

Fig. 3: Decoding tree for an (8, 4) polar code (image based on [12])

Initially, the log-likelihood ratios (LLRs) are loaded onto the root of the tree. Fig. 3(b) shows the data flow within the decoding tree. Each child node receives an LLR vector from its parent node, represented by $\boldsymbol{\alpha}$. Using these LLR values, the node computes $\boldsymbol{\alpha^l}$ and $\boldsymbol{\beta^l}$. Let N_s be the length of the polar code at stage s. Then, in [12] the min-sum approximation is used to calculate $\boldsymbol{\alpha^l}$:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{2} - 1\right\}:$$
$$\alpha_i^l = sign(\alpha_i) sign\left(\alpha_{i+\frac{N_s}{2}}\right) \min\left(|\alpha_i|, |\alpha_{i+\frac{N_s}{2}}|\right)$$

Once a leaf node is reached, a hard decision is made to estimate the codeword bit according to:

$$\hat{u}_i = h(\alpha) = \begin{cases} 0 & if\ \alpha \geq 0\ or\ i \in \mathcal{F} \\ 1 & otherwise \end{cases}$$

At the leaf node, $\boldsymbol{\beta} = \hat{u}_i$, and $\boldsymbol{\beta}$ is passed on to the parent node. The current node receives $\boldsymbol{\beta}$ as either $\boldsymbol{\beta^l}$ or $\boldsymbol{\beta^r}$ depending on the position of the child node as shown in Fig. 3(b). If $\boldsymbol{\beta^l}$ is received, then $\boldsymbol{\alpha^r}$ is calculated by the node as:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{2} - 1\right\}: \alpha_i^r = \left(1 - 2\beta_i^l\right)\alpha_i + \alpha_{i+\frac{N_s}{2}}$$

$\boldsymbol{\alpha^r}$ is sent to the child node on the right and it propagates through the tree until a hard decision is made at the leaves. When the current node receives $\boldsymbol{\beta^r}$, it sends $\boldsymbol{\beta}$ to its parent node by combining $\boldsymbol{\beta^l}$ and $\boldsymbol{\beta^r}$ as follows:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{2} - 1\right\}: \boldsymbol{\beta} = \left[\beta_i\ \beta_{i+\frac{N_s}{2}}\right] = \left[\beta_i^l \oplus \beta_i^r\ \ \beta_i^r\right]$$

III. Multi-Kernel Polar Codes

In [14] a method for the construction, encoding and decoding of polar codes with multiple kernels is proposed. The results demonstrated in the paper show that multi-kernel polar codes provide better error correction capabilities when compared to state-of-the art code puncturing and shortening schemes. To generate the frozen set, the authors suggest using a Monte-Carlo method, or the density evolution algorithm.

Once the frozen set is generated and the source-word is formed by placing the message bits in the information set bits, the source-word can be encoded first constructing a Tanner Graph like the one shown in Fig. 4.

In between each stage, there is a permutation matrix denoted by $\boldsymbol{P_i}$ which connects the outputs of stage $i-1$ to the inputs of stage i. The connections inside the T_2 block are the same as those shown in Fig. 1, and the connections inside the T_3 block is shown in Fig. 5. Also, the T_3 will be referred to as the ternary kernel.

The generator matrix for a multi-kernel polar code is defined in [14] to be: $G_N = T_{n_1} \otimes \dots \otimes T_{n_s}$. There are s stages from, and the length of the codeword is $N = n_1 \cdot \dots \cdot n_s$. Each stage has N/n_i blocks which each implements an T_{n_i} kernel. The rightmost stage is stage one, and the leftmost stage is stage s. Additionally, the codeword length corresponding to stage i is represented by N_i and is calculated as $N_i = \prod_{j=1}^{i-1} n_j$. The matrix $\boldsymbol{Q}$ is a permutation matrix called the canonical permutation, and $\boldsymbol{Q_i}$ is a permutation of N_{i+1} elements. The $\boldsymbol{Q}$ matrix is calculated as:

$$\boldsymbol{Q_i} = \begin{pmatrix} 1 & 2 & \dots & N_i & N_i+1 & N_i+2 & \dots & (n_i-1)N_i+1 & \dots & N_{i+1} \\ 1 & n_i+1 & \dots & (N_i-1)n_i+1 & 2 & n_i+2 & \dots & n_i & \dots & N_{i+1} \end{pmatrix}$$

For each i, such that $1 < i < s$, the $\boldsymbol{P_i}$ permutation matrix

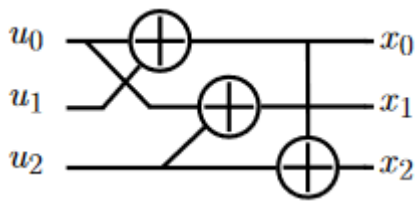

Fig. 5: XOR operations for encoding a polar code with the T_3 kernel and N = 3 (image based on [5])

can be calculated as:

$$\left(\boldsymbol{Q_i} \mid \boldsymbol{Q_i} + N_{i+1} \mid \boldsymbol{Q_i} + 2N_{i+1} \mid \dots \middle| \boldsymbol{Q_i} + \frac{N}{N_{i+1} - 1} N_{i+1}\right)$$

When $i = 1$, the $\boldsymbol{P_1}$ matrix is calculated as:

$$\boldsymbol{P_1} = (\boldsymbol{P_2} \cdot \dots \cdot \boldsymbol{P_s})^{-1}$$

And when $i = s$, the $\boldsymbol{P_s}$ permutation matrix is calculated as: $\boldsymbol{P_s} = \boldsymbol{Q_s}$

Once the Tanner graph is constructed, the source-word vector $\boldsymbol{u}$ is sent through it from left to right, to obtain the codeword $\boldsymbol{x}$.

A. SC Decoding for Multi-Kernel Polar Codes

The Successive Cancellation decoding algorithm, adapted for multi-kernel (MK) polar codes is presented in [14]. The tree representation of a MK polar code is shown in Fig. 6(a)

Like the decoding process for regular polar codes, the LLRs are loaded onto the root node of the decoding tree. The tree traversal algorithm now prioritizes the left node first, followed by the center node, and lastly the right node

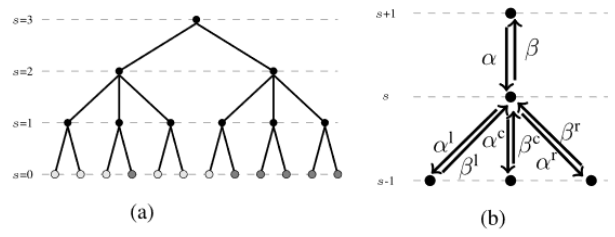

Fig. 6: Decoding tree for a P(12, 6) polar code with $G_6=T_2 \otimes T_3 \otimes T_2$

whenever a node corresponding to the T_3 kernel is encountered. Otherwise the priorities remain the same for the Arikan kernel.

The decoding of Arikan kernels follow the same equations as described before. In the case of ternary kernels, each child node receives an LLR vector from the parent node, denoted by $\boldsymbol{\alpha}$. The data flow for ternary kernel decoding is shown in Fig. 6(b). Using these LLR values, the node calculates $\boldsymbol{\alpha^l}$, $\boldsymbol{\alpha^c}$, and $\boldsymbol{\alpha^r}$. Representing the length of the polar code at stage s as N_s, the decoding equations for a ternary kernel are given in [14]. Using the min-sum approximation and $\varphi(x) = sign(x)$ the decoding equations are:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{3} - 1\right\}: \alpha_i^l = \varphi(\alpha_i)\varphi\left(\alpha_{i+\frac{N_s}{3}}\right)\varphi\left(\alpha_{i+\frac{2N_s}{3}}\right)\min\left(|\alpha_i|, \left|\alpha_{i+\frac{N_s}{3}}\right|, \left|\alpha_{i+\frac{2N_s}{3}}\right|\right)$$

At the leaf node, a hard decision is made like before to estimate the codeword bit:

$$\hat{u}_i = h(\alpha) = \begin{cases} 0 & if\ \alpha \geq 0\ or\ i \in \mathcal{F} \\ 1 & otherwise \end{cases}$$

These hard decisions are propagated back through the tree through the $\boldsymbol{\beta}$ vector. The parent node receives this vector as either $\boldsymbol{\beta^l}$, or $\boldsymbol{\beta^c}$, or $\boldsymbol{\beta^r}$, depending on whether it is being returned from the left, center, or right child node.

From $\boldsymbol{\beta^l}$, the $\boldsymbol{\alpha_c}$ vector is constructed as:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{3} - 1\right\}: \alpha_i^c = \left(1 - 2\beta_i^l\right)\alpha_i + \varphi(\alpha_{i+\frac{N_s}{3}})\varphi(\alpha_{i+\frac{2N_s}{3}})\min\left(|\alpha_{i+\frac{N_s}{3}}|, |\alpha_{i+\frac{2N_s}{3}}|\right)$$

This $(\boldsymbol{\alpha_c})$ vector is sent down to the center child and the current node waits until it receives $\boldsymbol{\beta_c}$ from the center child node. Lastly, $\boldsymbol{\alpha^r}$ is calculated by:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{3} - 1\right\}: \alpha_i^r = \left(1 - 2\beta_i^l\right)\alpha_{i+\frac{N_s}{3}} + \left(1 - 2\beta_i^l \oplus \beta_i^l\right)\alpha_{i+\frac{2N_s}{3}}$$

Upon receiving $\boldsymbol{\beta^r}$ from the right child node, the current node combines all the $\boldsymbol{\beta^l}$, $\boldsymbol{\beta^c}$ and $\boldsymbol{\beta^r}$ vectors to produce the $\boldsymbol{\beta}$ and sends it to its parent node:

$$\forall\, i \in \left\{0, 1, \dots, \frac{N_s}{2} - 1\right\}: \boldsymbol{\beta} = \left[\beta_i\ \beta_{i+\frac{N_s}{3}}\ \beta_{i+\frac{2N_s}{3}}\right] = \left[\beta_i^l \oplus \beta_i^c\ \ \beta_i^l \oplus \beta_i^r\ \ \beta_i^c \oplus \beta_i^r\right]$$

This process continues until all the leaf nodes have been decoded.

B. Frozen Set Construction for MK Polar Codes

In [1] the indices of the frozen set are calculated by using the Bhattacharya parameter expansion algorithm, also described in [11]. This method is only proven to be exact for the Binary Erasure Channel (BEC). A more accurate representation of the bit reliabilities in an AWGN channel is described in [15]. This method is known as Gaussian Approximation. In [5] this method is expanded to Multi-Kernel polar codes.

The Gaussian Approximation method, as described in [5] is as follows. Initially, for an AWGN channel with zero mean and variance of σ^2 all N reliabilities of indices are initialized as:

$$z_i^N = \frac{2}{\sigma^2} = 4R\frac{E_b}{N_0}$$

For the indices of the next stage, if the next stage corresponds to an Arikan kernel, use the following equations to update the reliabilities of the bits:

$$z_{2i}^{\frac{N}{2}} = 2z_i^N,$$

$$z_{2i-1}^{\frac{N}{2}} = \phi^{-1}\left(1 - \left(1 - \phi(z_i^N)\right)^2\right)$$

If the indices correspond to the ternary kernel, use the following equations to update reliabilities:

$$z_{3i}^{\frac{N}{3}} = 2z_i^N$$

$$z_{3i-1}^{\frac{N}{3}} = \phi^{-1}\left(1 - \left(1 - \phi(z_i^N)\right)^2\right) + z_i^N$$

$$z_{3i-2}^{\frac{N}{3}} = \phi^{-1}\left(1 - \left(1 - \phi\left(\phi^{-1}\left(1 - \left(1 - \phi(z_i^N)\right)^2\right)\right)\right)\left(1 - \phi(z_i^N)\right)\right)$$

Where $\phi(x)$ and $\phi^{-1}(x)$ are approximated as:

$$\phi(x) = \begin{cases} e^{0.0564x^2 - 0.485x} & if\ x < 0.8678 \\ e^{\alpha x^\gamma + \beta} & otherwise \end{cases}$$

$$\phi^{-1}(x) = \begin{cases} 4.3049\left(1 - \sqrt{1 + 0.9567 \cdot \ln(x)}\right) & if\ x > 0.6846 \\ (a \cdot \ln(x) + b)^c & otherwise \end{cases}$$

The constants are: $\alpha = -0.4527$, $\beta = 0.0218$, $\gamma = 0.86$, $a = \frac{1}{\alpha}$, $b = \frac{-\beta}{\alpha}$ and $c = \frac{1}{\gamma}$. The exact formulas for these equations are given in [5].

All the frozen set design method shown in [11] and in [5] consider the reliability of the bit-indices as determined due to the polarization effect. However, in [16] it is shown that for short block lengths, the minimum distance of the code is far more important than the polarization effect when using a SCL decoder. As a result, a method is described for the construction of the frozen set in order to maximize the distance of the multi-kernel polar code.

In [16] the minimum distance spectrum of a generator matrix G_N is defined as the vector containing the maximum minimum-distance achievable by a code of length N. It is proven that for a generator in the form $G_N = T_2^{\otimes n} \otimes T_p$, where T_p is any kernel of size $p \times p$, and $p \neq 2$, the minimum distance spectrum is given as:

$$S_{G_N} = sort(S_{T_2^{\otimes n}} \otimes S_{T_p})$$

$sort(x)$ is the vector x sorted in descending order, S_{T_p} is the minimum distance spectrum of the T_p kernel, and $S_{T_2^{\otimes n}}$ is the minimum distance spectrum of the $T_2^{\otimes n}$ kernel. For polar codes, $S_{T_2^{\otimes n}} = sort\left([2\ \ 1]^{\otimes n}\right)$. It is suggested that S_{T_p} be calculated through an exhaustive search algorithm. A pseudocode implementation of such an algorithm is described in Algorithm 1.

```
foreach k in [1:K]:
   max_min_dist = 0

   foreach combination of k rows in Tp:
      G = submatrix of Tp consisting of k selected rows
      min_dist = N+1
      min_dist_order = ∅

      foreach data in [1:2^k]:
         codeword = data * G

         if(weight(codeword) < min_dist):
            min_dist = weight(codeword)
         endif
      endforeach

      if(min_dist > max_min_dist):
         max_min_dist = min_dist
         min_dist_order = combination
      endif
   endforeach

   S_Tp[k – 1] = max_min_dist
   I[k – 1] = min_dist_order
endforeach
```

Algorithm 1: find_min_dist_spect_T_p()

Additionally, the list $I^k = \{i_1^k, i_2^k, \dots, i_k^k\}$ is associated to each entry in S_{T_p} to store the k row indices that achieves the maximum distance. Once S_{G_N} is calculated, the row indices of G_N that will generate the minimum-distance are selected through a Greedy Row-Selection Algorithm.

The Greedy Row-Selection algorithm starts with the generating $r_N = (2,1)^{\otimes n} \otimes S_{T_p}$. At each loop of the algorithm, index of the maximum element in r_N is identified and represented by l. Then, maximum the row index of the corresponding row in the T_p matrix is selected and represented by c, and the row index of the row in the generator matrix is selected and represented by q. This loop repeats K times in order to find the K most reliable indices which make up the information set. The remaining N – K indices make up the frozen set. A pseudocode implementation of the Greedy Row-Selection algorithm is shown in Algorithm 2.

```
foreach x in [1:K]:
   ℐ = ∅          // Initialize information set as empty set
   Load I
   r_N = calculate_rN()

   l = argmax(r_N)
   c = l mod p
   q = l − c
   r_N[l] = 0

   if(c > 0 AND (ℐ ≠ ∅):
      Remove all elements from I[c − 1] + q in ℐ
   endif

   Add all elements from I[c] + q to ℐ

endforeach
```

Algorithm 2: Greedy Row Selection Algorithm to generate Information Set

C. Fast-SSC Decoding of MK Polar Codes

The objective of the Fast-SSC decoding algorithm, as presented in [5] is to prune the decoding tree and identify nodes for which specialized decoders can be used to decode them with a lower latency. These nodes are detected based on the pattern of the frozen set associated with their leaf nodes. In other words, starting at the node, if a downwards tree traversal is performed, then the indices of the leaf nodes are compared to the indices within the frozen set. Depending on a match on the pattern of frozen bits within the leaves, the node is identified as either a regular node, or one of the special nodes. Currently, there are four types of nodes that have been identified.

1. Rate-0 Node:

In a Rate-0 Node, all the indices of the bits associated with that node are in the frozen set. As a result, all the bits can be decoded as zero, instead of traversing through the entire tree.

2. Rate-1 Node:

In a Rate-1 Node, none the indices of the bits associated with that node are in the frozen set. As a result, instead of propagating the LLR vector down the tree, a hard decision can be made on the LLR vector ($\boldsymbol{\alpha}$) at the Rate-1 node itself to find the $\boldsymbol{\beta}$ vector. Propagating the $\boldsymbol{\beta}$ down the tree results in all the associated bits being decoded. The difference now is the

decoding is no longer serial in nature since all the nodes can be decoded independently.

3. Single-Parity Check Node:

For node with n bits associated with it and if the leftmost bit associated bit is a frozen bit, then the node is a single-parity check node. To decode this node, a hard decision is made on the LLR vector ($\boldsymbol{\alpha}$) to get the $\boldsymbol{\beta}$ vector. Then the parity of $\boldsymbol{\beta}$ is computed as the bitwise-XOR of the vector. If the parity bit is 1, then the bit at the least reliable index is flipped. Lastly, the $\boldsymbol{\beta}$ vector is propagated down the tree in the same manner as the Rate-1 Node to decode the individual bits.

4. Repetition Node:

For a node where there is only one associated bit in the information set, and if all nodes below the node in question have only two children, and if the rightmost associated bit is the information bit, then this node is a Repetition Node. In particular a REP2 node. All elements of the $\boldsymbol{\beta}$ vector are the same and can be calculated to be:

$$\beta[i] = h\left(\sum_{j} \alpha[j]\right)$$

In the case of ternary kernels, there are three sub-types of repetition nodes: REP3A, REP3B and REP3C [5].

IV. Software Model

Simulation of multi-kernel polar code construction, encoding and decoding is done through a software model developed using C++11. The program takes the following parameters as input:

- Block length (N)
- Message length (K)
- Generator Kernel Order: This is represented by an array of integers where each integer represents size of the polarizing kernel. Currently, only Arikan and Ternary kernels are supported
- Starting SNR: The initial SNR in dB of the AWGN channel to be simulated
- Ending SNR: the final SNR in dB of the AWGN channel to be simulated
- SNR Step Size: size of each SNR increment in dB
- Number of frame errors to simulate till, for each SNR
- Code Construction method: Minimum distance construction as described in [16] or the Gaussian Approximation construction as described in [5]
- Decoding Method: Successive Cancellation decoding as described in [1] or Fast-SSC decoding as described in [5]

Additionally, there are options internal to the program that can be changed to control the data type representations of bits and real numbers. The real numbers are used to store the received vector and the LLR values at each step. Lastly, the encoding is done through the Tanner Graph as described in [14].

When the software model is run, it shows information on the frozen set bit indices, location and type of specialized node detected, the number of nodes before and after pruning the tree, and the percentage reduction in tree size. After this, it starts the simulation as specified by the input parameters. For each channel SNR value to be simulated, it shows the number of frames simulated (FRA), number of bit errors (BE), number of frame errors (FE), the bit error rate (BER) and the frame error rate (FER). The SNR and the BER can then be plotted using a tool such as MATLAB.

A. Observations

For an N = 144, K=72, polar code constructed using the minimum distance construction and decoded using the Fast-SSC algorithm, the software model shows a reduction 54.5% in decoding tree size (from 222 nodes in the SC decoding tree, to 101 nodes in the Fast-SSC tree). Plotting the Bit Error Rate against the Channel SNR, Fig. 7 is obtained.

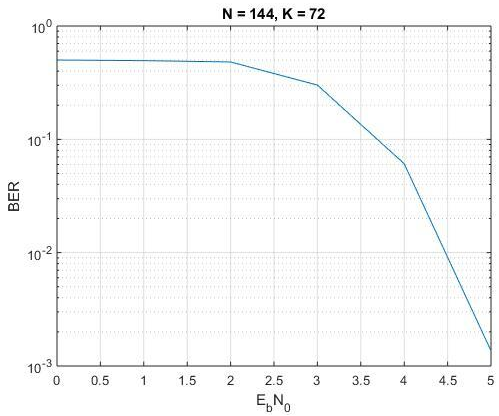

Fig. 7: Error Correction Performance of an P(144, 72) code

	%Reduction Gaussian Approx.	%Reduction Min. Dist.
N=96, K=24	84.18 %	51.90 %
N=96, K=48	81.64 %	50 %
N=96, K=72	84.18 %	72.78 %
N=144, K=36	87.28 %	61.26 %
N=144, K=72	82.88 %	54.50 %
N=144, K=108	85.13 %	69.82 %

Table 1: Latency reduction of various block lengths and code rates

For the same P(144, 72) code but now constructed using the Gaussian Approximation method, the software model shows a tree size reduction of 82.9% (from 222 nodes to 38).

It can be observed from Table 1 that Gaussian Approximation yields better latency reduction when decoded using a Fast-SSC decoder. For both the Gaussian Approximation and Minimum Distance construction, there is a greater reduction of tree size when the code rate is not 1/2. This can be explained since, when there is a larger number of information bits compared to frozen bits, its more likely that there will be only one frozen bit in a sub-code and most of the

remaining bits are information bits. This leads to the existence of more SPC nodes. Similarly, when there is a larger number of frozen bits, its more likely that there will be only one information bit in a sub-code and most of the remaining bits are frozen bits, which give rise to more Repetition nodes.

V. Conclusion

In this work, we have summarized various methods of code construction, encoding and decoding of both multi-kernel polar codes, and polar codes consisting solely of the Arikan Kernel. The software model presented in this paper should act as a reference for a hardware implementation of Multi-Kernel Polar Code decoders using the Fast-SSC algorithm.

References

[1] E. Arikan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," in IEEE Transactions on Information Theory, vol. 55, no. 7, pp. 3051-3073, July 2009.

[2] 3GPP, “NR; Multiplexing and Channel Coding,” http://www.3gpp.org/DynaReport/38-series.htm, Tech. Rep. TS 38.212, June 2018, Release 15.

[3] A. Alamdar-Yazdi and F. R. Kschischang, "A Simplified Successive-Cancellation Decoder for Polar Codes," in IEEE Communications Letters, vol. 15, no. 12, pp. 1378-1380, December 2011.

[4] G. Sarkis, P. Giard, A. Vardy, C. Thibeault and W. J. Gross, "Fast Polar Decoders: Algorithm and Implementation," in IEEE Journal on Selected Areas in Communications, vol. 32, no. 5, pp. 946-957, May 2014.

[5] A. Cavatassi, T. Tonnellier and W. J. Gross, "Fast Decoding of Multi-Kernel Polar Codes," 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 2019, pp. 1-6.

[6] I. Tal and A. Vardy, "List Decoding of Polar Codes," in IEEE Transactions on Information Theory, vol. 61, no. 5, pp. 2213-2226, May 2015.

[7] S. B. Korada, E. Şaşoğlu and R. Urbanke, "Polar Codes: Characterization of Exponent, Bounds, and Constructions," in IEEE Transactions on Information Theory, vol. 56, no. 12, pp. 6253-6264, Dec. 2010.

[8] K. Niu, K. Chen and J. Lin, "Beyond turbo codes: Rate-compatible punctured polar codes," *2013 IEEE International Conference on Communications (ICC)*, Budapest, 2013, pp. 3423-3427.

[9] V. Miloslavskaya, "Shortened Polar Codes," in IEEE Transactions on Information Theory, vol. 61, no. 9, pp. 4852-4865, Sept. 2015.

[10] R. Wang and R. Liu, "A Novel Puncturing Scheme for Polar Codes," in *IEEE Communications Letters*, vol. 18, no. 12, pp. 2081-2084, Dec. 2014.

[11] H. Vangala, E. Viterbo, and Y. Hong,“A comparative study of polar code constructions for the AWGN channel,” in arXiv preprint arXiv:1501.02473, 2015

[12] G. Coppolino, C. Condo, G. Masera and W. J. Gross, "A Multi-Kernel Multi-Code Polar Decoder Architecture," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4413-4422, Dec. 2018.

[13] M. P. C. Fossorier, M. Mihaljevic and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," in *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673-680, May 1999.

[14] F. Gabry, V. Bioglio, I. Land and J. Belfiore, "Multi-kernel construction of polar codes," *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, Paris, 2017, pp. 761-765.

[15] P. Trifonov, “Efficient Design and Decoding of Polar Codes,”IEEE Trans. on Commun., vol. 60, no. 11, pp. 3221–3227, nov 2012.

[16] V. Bioglio, F. Gabry, I. Land and J. Belfiore, "Minimum-Distance Based Construction of Multi-Kernel Polar Codes," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Singapore, 2017, pp. 1-6.