Trellis-based Decoding Techniques for Polar Codes with Large Kernels

Peter Trifonov
ITMO University, Russia
Email: pvtrifonov@corp.ifmo.ru

***Abstract*—A recursive trellis-based algorithm is proposed for computing the probabilities arising in the successive cancellation (SC) decoding algorithm for polar codes with arbitrary linear kernels. This approach enables one to re-use intermediate values obtained at different phases. Arikan SC algorithm is shown to be a special case of this method.**

I. Introduction

Polar codes are a novel class of capacity-achieving codes with low-complexity construction, encoding and decoding algorithms [1]. However, classical polar codes have very high scaling exponent [2]. Therefore, obtaining reasonable performance at moderate code length requires one to employ various extensions of polar codes, such as polar codes with CRC and polar subcodes, together with list decoding techniques [3], [4]. However, employing list decoding (at least, its simplified version, which admits theoretical analysis) does not change the scaling exponent of polar codes [5]. In practice it appears that, for polar codes with CRC and polar subcodes of length more than a few thousands, SC list (SCL) decoding with very large list size is needed to obtain competitive performance.

Replacing the 2×2 matrix in the Arikan construction with larger matrices allows one to obtain higher rate of polarization [6]. Polar codes with large kernels were shown to provide asymptotically optimal scaling exponent [7]. However, the decoding complexity of polar codes with large kernels is, in general, prohibitively high, except for the case of very small kernels [8]. Unfortunately, obtaining polarization rate greater than $1/2$ requires employing a kernel of dimension at least 15 in the binary case [6]. Therefore, polar codes with larger kernels are commonly believed to be impractical.

It was shown in [9] that, by a careful joint design of a kernel and decoding (processing) algorithm, it is possible to obtain kernels, such that the corresponding polar (sub)codes under SCL decoding require lower number of arithmetic operations compared to similar codes with Arikan kernel. However, construction of the corresponding processing algorithms is, in general, a very difficult task.

In this paper we present a generic technique, which may enable reduced complexity processing of large kernels. The proposed approach relies on the trellis representation of the codes generated by submatrices of the considered kernels. Furthermore, it may lead to new constructions of kernels with good polarization properties and low processing complexity.

II. Background

A. Polar codes

Let F_l be an $l \times l$ non-singular matrix over $\mathbb{F}_2$, such that no column permutation can transform it into an upper triangular matrix [6]. A polarizing transformation is given by $A = B_{l,m}F^{\otimes m}$, where $B_{l,m}$ is a digit-reversal permutation matrix, corresponding to mapping $\sum_{i=0}^{m-1} t_i l^i \rightarrow \sum_{i=0}^{m-1} t_{m-1-i} l^i$,$t_i \in [l]$, and $[l] = \{0, 1, , \ldots, l-1\}$. This construction can be also extended to the case of mixed-kernel polarizing transformations [10]. It is possible to show that a binary input memoryless symmetric channel $\mathcal{W}(y|c) = \mathcal{W}_0^{(0)}(y|c)$ together with matrix A gives rise to $n = l^m$ bit subchannels

$$\mathcal{W}_m^{(i)}(y_0^{n-1}, u_0^{i-1}|u_i) = \frac{1}{2^{n-1}} \sum_{u_{i+1}^{n-1} \in \mathbb{F}_2^{n-i-1}} \prod_{j=0}^{n-1} \mathcal{W}_0^{(0)}(y_j|(u_0^{n-1}A)_j) \quad (1)$$

with capacities approaching 0 or 1 symbols per channel use, and fraction of noiseless subchannels approaching $I(\mathcal{W})$, the capacity of channel $\mathcal{W}$. An $(n = l^m, k)$ polar code over $\mathbb{F}_2$ is a set of codewords $c_0^{n-1} = u_0^{n-1}A$, where $u_i = 0, i \in \mathcal{F}$, $\mathcal{F} \subset [n]$ is the set of frozen symbol indices, and $|\mathcal{F}| = n - k$. The set $\mathcal{F}$ is typically selected as the set of indices i of low-capacity subchannels $\mathcal{W}_m^{(i)}$.

It is convenient to define probabilities

$$\mathcal{W}_m^{(i)}\left\{u_0^i|y_0^{l^m-1}\right\} = \mathcal{W}_m^{(i)}\left\{U_0^i = u_0^i|Y_0^{l^m-1} = y_0^{l^m-1}\right\} = \frac{\mathcal{W}_m^{(i)}(y_0^{n-1}, u_0^{i-1}|u_i)P\{u_i\}}{\mathcal{W}(y_0^{n-1})}, \quad (2)$$

where $P\{u_i\} = 1/2, u_i \in \{0, 1\}$ is the probability distribution of input symbols, and $W(y_0^{n-1})$ is the probability density function of the channel output. These probabilities can be recursively computed as

$$\mathcal{W}_\lambda^{(lj+i)}\left\{u_0^{lj+i}|y_0^{N-1}\right\} = \sum_{u_{lj+i+1}^{lj+l-1}} \prod_{s=0}^{l-1} \mathcal{W}_{\lambda-1}^{(j)}\left\{(u_{lt}^{lt+l-1}F_l)_s, 0 \le t \le j|y_{\frac{N}{l}s}^{\frac{N}{l}s+\frac{N}{l}-1}\right\}, \quad (3)$$

where $i \in [l], j \in [\frac{N}{l}]$, $N = l^\lambda$. The SC decoding algorithm makes decisions

$$\hat{u}_i = \begin{cases} 0 & i \in \mathcal{F} \\ \arg\max_{u_i} \mathcal{W}_m^{(i)}(\hat{u}_0^{i-1}, u_i|y_0^{n-1}), & i \notin \mathcal{F}. \end{cases}$$

By re-using the intermediate values $\mathcal{W}_{\lambda-1}^{(j)}$, one can implement SC decoding with complexity $O(n\log n)$ operations of computing (3). However, the cost of straightforward evaluation of (3) is $O(l2^l)$.

One way to simplify these calculations is to employ an approximation

$$\mathcal{W}_m^{(i)}(u_0^i|y_0^{n-1}) \approx W_m^{(i)}(u_0^i|y_0^{n-1}) = \max_{u_{i+1}^{n-1}\in\mathbb{F}_2^{n-i-1}} \prod_{j=0}^{n-1} \mathcal{W}_0^{(0)}((u_0^{n-1}A)_j|y_j).$$

These probabilities can be recursively computed as

$$W_\lambda^{(lj+i)}\left\{u_0^{lj+i}|y_0^{N-1}\right\} = \max_{u_{lj+i+1}^{lj+l-1}} \prod_{s=0}^{l-1} W_{\lambda-1}^{(j)}\left\{(u_{lt}^{lt+l-1}F_l)_s, 0\le t\le j|y_{\frac{N}{l}s}^{\frac{N}{l}s+\frac{N}{l}-1}\right\}, \quad (4)$$

where $W_0^{(0)}(c|y)=\mathcal{W}(c|y)$. One application of this operation will be referred to as *kernel processing*. Integer i will be referred to as the *phase* of the kernel processor. The maximization in (4) can be recognized as decoding in a coset of the code $\mathcal{C}^{(i)}$ generated by rows $i+1,\dots,l-1$ of kernel F_l, where the coset representative is given by a linear combination of rows $0,\dots,i$ of F_l with coefficients u_{lj}^{lj+i}. In the case of symmetric channels decoding in a coset of $\mathcal{C}^{(i)}$ can be implemented by employing a decoder of $\mathcal{C}^{(i)}$ with appropriately adjusted input LLRs. Therefore, for the sake of simplicity we will assume $u_0^{lj+i-1}=0$ in what follows. The codes $\mathcal{C}^{(i)}$ may have some structure, which may be used to reduce the complexity of their decoding, i.e. solving the optimization problem (4).

For brevity, we will consider below only the case of $m=1$.

B. Recursive trellis decoding of binary linear codes

A recursive trellis-based decoding algorithm for linear codes was proposed in [11]. Given a linear code C, let $C_{h,h'}$ be its subcode, such that all its codewords have non-zero symbols only in positions $h\le i<h'$. Let $p_{h,h'}(C)$ be a linear code obtained by puncturing all symbols, except those in positions $h\le i<h'$, from codewords of C. Let us further define $s_{h,h'}(C)=p_{h,h'}(C_{h,h'})$, i.e. a code obtained from C by shortening it on all symbols except those with indices $h\le i<h'$. $s_{h,h'}(C)$ and $p_{h,h'}(C)$ will be referred to as *section codes*. Consider a minimal trellis of code C, and its section corresponding to symbols from x to y. It is possible to show that the paths between two adjacent states in this section correspond to a coset in $p_{x,y}(C)/s_{x,y}(C)$. Furthermore, this coset may appear multiple times. This enables one to reduce the complexity of maximum likelihood (ML) decoding C by pre-computing the metrics of these paths. That is, for each coset $D\in p_{x,y}(C)/s_{x,y}(C)$ one needs to identify the most probable element $l(D)$, and store its probability $m(D)$. Let the *composite branch table* $CBT_{x,y}$ be a data structure storing the pairs $(l(D),m(D))$. In the case of conventional decoding of some (n,k) code, $p_{0,n}(C)/s_{0,n}(C)$ contains a single element, so $CBT_{0,n}$ contains one entry, which corresponds to a solution of the ML decoding problem.

The straightforward approach to construction of a composite branch table for some code C is to enumerate all codewords of $p_{x,y}(C)$, and find the most probable one for each coset in $p_{x,y}(C)/s_{x,y}(C)$. However, more efficient approach was suggested in [11] for the case of $y-x\ge 2$.

Consider some $D_y\in p_{x,y}(C)/s_{x,y}(C)$, and let $S_z(D_y)=\{D_z\in p_{x,z}(C)/s_{x,z}(C)|D_z\subseteq p_{x,z}(D_y)\}$ for some $z: x\le z<y$, where $p_{x,z}(D_y)$ is a truncation of D_y on positions from x to z. Observe that for any $D_z\in S_z(D_y)$ there is exactly one coset $\mathrm{adj}(D_z,D_y)\in p_{z,y}(C)/s_{z,y}(C)$, such that $D_z\circ\mathrm{adj}(D_z,D_y)\subset D_y$, where $\circ$ denotes the concatenation operator.

Let us further consider a two-section trellis for $p_{x,y}(C)/s_{x,y}(C)$, such that:

1) It has a single initial state $\sigma_{x,0}$ at time x.
2) There is a one-to-one correspondence between states $\sigma_{i,D}$ at time $i\in\{z,y\}$ and cosets $D\in p_{x,i}(C)/s_{x,i}(C)$.

Then the branch label between states $\sigma_{x,0}$ and σ_{z,D_z} corresponds to D_z, and branch label between σ_{z,D_z} and σ_{y,D_y} corresponds to $\mathrm{adj}(D_z,D_y)$.

Given $CBT_{x,z}(C)$ and $CBT_{z,y}(C)$ for some $x<z<y$, one can construct $CBT_{x,y}(C)$ as a set of pairs $(l(D_y),m(D_y))$, $D_y\in p_{x,y}(C)/s_{x,y}(C)$, where

$$m(D_y)=\max_{D_z\in S_z(D_y)}(m(D_z)\cdot m(\mathrm{adj}(D_z,D_y))). \quad (5)$$

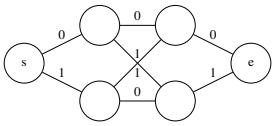
The corresponding elements $l(D_y)$ are obtained as a concatenation of D_z and $\mathrm{adj}(D_z,D_y)$, which deliver the maximum in this expression. This enables recursive construction of CBTs.

III. Trellis-based kernel processing

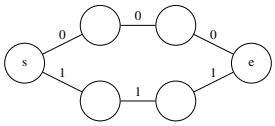
A. Extended trellises of kernel codes

Polar codes can be considered as a result of recursive application of the construction of generalized concatenated codes [12], [13]. These codes rely on non-systematic inner codes, as well as the corresponding soft-decision decoding algorithms. An optimal soft-input soft-output decoding algorithm for non-systematically encoded linear block codes was presented in [14]. This algorithm can be easily tailored to implement computation of (3) or (4) . To do this, let us consider an extended $(l+1,l-i)$ code $\overline{\mathcal{C}}^{(i)}$ generated by matrix $G^{(i)}$. First l columns of this matrix are obtained by taking $l-i$ last rows of kernel F_l. The last column has 1 in the 0-th row, and zeroes in the remaining position. Then computing (4) for $u_i\in\mathbb{F}_2$ is equivalent to finding the most probable codewords of code $\overline{\mathcal{C}}^{(i)}$ having 0 and 1 in the last symbol. This can be implemented by running the Viterbi algorithm over the trellis of $\overline{\mathcal{C}}^{(i)}$, assuming that the last codeword symbol is erased. Such trellis will be referred to as an extended trellis of $\mathcal{C}^{(i)}$.

Example 1. *Consider the case of Arikan kernel* $F_2=\begin{pmatrix}1&0\\1&1\end{pmatrix}$. *One obtains* $G^{(0)}=\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}$ *and* $G^{(0)}=$



(a) An extended trellis for $\overline{\mathcal{C}}^{(0)}$



(b) An extended trellis for $\overline{\mathcal{C}}^{(1)}$

Fig. 1: Extended trellises for the Arikan kernel

$\begin{pmatrix}1 & 1 & 1\end{pmatrix}$. *The corresponding extended trellises are shown in Figure 1.*

B. Recursive trellis processing

The recursive trellis decoding algorithm described in Section II-B can be immediately applied to the kernel processing problem. In this case, due to invertibility of F_l, one has $|p_{0,l}(\overline{\mathcal{C}}^{(i)})/s_{0,l}(\overline{\mathcal{C}}^{(i)})| = 2$, so the composite branch table of $\overline{\mathcal{C}}^{(i)}$, denoted $CBT_{0,l}^{(i)}$, contains exactly $W_1^{(i)}\left\{u_0^i|y_0^{l-1}\right\}, u_i \in \mathbb{F}_2$. Observe that in the case of kernel processing we do not actually need to store $l(D)$, since only the probabilities $m(D)$ have to be computed.

However, the factorsets $p_{x,y}(\overline{\mathcal{C}}^{(i)})/s_{x,y}(\overline{\mathcal{C}}^{(i)})$ arising in this algorithm may be identical for different i. This enables one to re-use the obtained intermediate results, obtaining thus additional complexity savings.

Example 2. *Consider the 2-iterated Arikan kernel* $F_4 = \begin{pmatrix}1 & 0 & 0 & 0\\ 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 0\\ 1 & 1 & 1 & 1\end{pmatrix}$. *It can be seen that* $p_{0,4}(\overline{\mathcal{C}}^{(0)})$ *is a* $(4,4,1)$ *code, while* $s_{0,4}(\overline{\mathcal{C}}^{(0)})$ *is a* $(4,3,2)$ *code. Hence, we need to consider cosets of the latter code given by vectors* $(0,0,0,0)$ *and* $(1,0,0,0)$. *Furthermore,* $p_{0,2}(\overline{\mathcal{C}}^{(0)})$ *and* $p_{2,4}(\overline{\mathcal{C}}^{(0)})$ *are* $(2,2,1)$ *codes, while* $s_{0,2}(\overline{\mathcal{C}}^{(0)})$ *and* $s_{2,4}(\overline{\mathcal{C}}^{(0)})$, *are* $(2,1,2)$ *codes. Therefore*[1], $S_2(0,0,0,0) = \{(0,0)\}$ *and* $S_2(1,0,0,0) = \{(1,0)\}$. *Observe that the elements of these sets, which correspond to the states of the two-section trellis at time 2, are exactly the coset representatives of the* $(2,1,2)$ *code. Hence, one obtains*

$$CBT_{0,2}^{(0)} = [\max(w_{00}w_{01}, w_{10}w_{11}), \max(w_{10}w_{01}, w_{00}w_{11})]$$

and

$$CBT_{2,4}^{(0)} = [\max(w_{02}w_{03}, w_{12}w_{13}), \max(w_{12}w_{03}, w_{02}w_{13})],$$

where $w_{ij} = W(c = i|y_j)$. *The probabilities required by phase-0 kernel processing are obtained as*

$$\begin{aligned}CBT_{0,4}^{(0)} = [&\\ &\max(CBT_{0,2}^{(0)}[0]CBT_{2,4}^{(0)}[0], CBT_{0,2}^{(0)}[1]CBT_{2,4}^{(0)}[1]),\\ &\max(CBT_{0,2}^{(0)}[0]CBT_{2,4}^{(0)}[1], CBT_{0,2}^{(0)}[1]CBT_{2,4}^{(0)}[0])]\end{aligned}$$

where $CBT_{x,y}^{(0)}[D]$ *is the* D*-th entry in the corresponding CBT.*

It can be also seen that $p_{x,x+2}(\overline{\mathcal{C}}^{(1)}) = p_{x,x+2}(\overline{\mathcal{C}}^{(0)})$ *and* $s_{x,x+2}(\overline{\mathcal{C}}^{(1)}) = s_{x,x+2}(\overline{\mathcal{C}}^{(0)})$ *for* $x \in \{0,2\}$. *However,* $p_{0,4}(\overline{\mathcal{C}}^{(1)})$ *is a* $(4,3,2)$ *code, while* $s_{0,4}(\overline{\mathcal{C}}^{(1)})$ *is a* $(4,2,2)$ *code generated by two last rows of* F_4. *Hence, we need to consider the cosets of the latter codes given by vectors* $(0,0,0,0)$ *and* $(1,0,1,0)$. *Hence, assuming that* $u_0 = 0$, *one obtains*

$$CBT_{0,4}^{(1)} = [CBT_{0,2}^{(0)}[0]CBT_{2,4}^{(0)}[0], CBT_{0,2}^{(0)}[1]CBT_{2,4}^{(0)}[1]].$$

If $u_0 = 1$, *the set of coset representatives considered at this phase would need to be changed to* $(1,0,0,0)$ *and* $(0,0,1,0)$.

At phase 2 *the codes* $p_{0,2}(\overline{\mathcal{C}}^{(2)})$ *and* $p_{2,4}(\overline{\mathcal{C}}^{(2)})$ *are* $(2,1,2)$ *codes, while* $s_{0,2}(\overline{\mathcal{C}}^{(2)})$ *and* $s_{2,4}(\overline{\mathcal{C}}^{(2)})$ *are trivial* $(2,0,\infty)$ *codes containing only zero codeword. Code* $p_{0,4}(\overline{\mathcal{C}}^{(2)})$ *is a* $(4,2,2)$ *code, while* $s_{0,4}(\overline{\mathcal{C}}^{(2)})$ *is a* $(4,1,4)$ *code. Hence, one needs to consider the cosets of the latter code given by vectors* $(0,0,0,0)$ *and* $(1,1,0,0)$, *so that* $S_2(0,0,0,0) = \{(0,0)\}$ *and* $S_2(1,1,0,0) = \{(1,1)\}$. *That is, we need to consider only cosets of the* $(2,0,\infty)$ *code given by* $(0,0)$ *and* $(1,1)$.

Therefore, in the case of $u_0 = u_1 = 0$ *one obtains*

$$CBT_{0,2}^{(2)} = [w_{00}w_{01}, w_{10}w_{11}], CBT_{2,4}^{(2)} = [w_{02}w_{03}, w_{12}w_{13}]$$

and

$$\begin{aligned}CBT_{0,4}^{(2)} = [&\\ &\max(CBT_{0,2}^{(2)}[0]CBT_{2,4}^{(2)}[0], CBT_{0,2}^{(2)}[1]CBT_{2,4}^{(2)}[1]),\\ &\max(CBT_{0,2}^{(2)}[0]CBT_{2,4}^{(2)}[1], CBT_{0,2}^{(2)}[1]CBT_{2,4}^{(2)}[0])].\end{aligned}$$

Similarly,

$$CBT_{0,4}^{(3)} = [CBT_{0,2}^{(2)}[0]CBT_{2,4}^{(2)}[0], CBT_{0,2}^{(2)}[1]CBT_{2,4}^{(2)}[1]].$$

The above calculations can be readily recognized as a max-product implementation of the SC algorithm for Arikan polar codes. Observe that they can be further simplified by going to the LLR domain, i.e. by employing the min-sum algorithm. However, we do not consider these tricks for the sake of clarity.

Hence, the complexity of the recursive trellis processing method depends on:

- the partitioning strategy, i.e. rules for selection of z for any given pair x, y;
- number of distinct codes $p_{x,y}(\overline{\mathcal{C}}^{(i)})$, $s_{x,y}(\overline{\mathcal{C}}^{(i)}), 0 \le i < m$, for each pair x, y obtained for the considered partitioning strategy, i.e. the number of times one can re-use the intermediate results from previous phases;
- the complexity of computing (5).

It was shown in [11] that the latter quantity is given by

$$|S_z(D_y)| = \frac{|s_{x,y}(\overline{\mathcal{C}}^{(i)})|}{|s_{x,z}(\overline{\mathcal{C}}^{(i)})||s_{z,y}(\overline{\mathcal{C}}^{(i)})|}. \tag{6}$$

Note that a trellis sectionalization method, which is optimal for standalone decoding of some code, may not be optimal in the considered scenario, since here one needs to take into account the possibility for re-using the intermediate decoding results at various phases i.

[1]Strictly speaking, the elements of $S_z(D)$ are cosets of $s_{0,z}(\overline{\mathcal{C}}^{(i)})$. We represent each coset by some vector drawn from it.

IV. Examples of application

A. Iterated Arikan kernel

In this section we show that the Arikan SC decoding algorithm is a special case of the recursive decoding algorithm introduced in [11]. To ensure consistent notation, we state this result in terms of processing of the m-iterated Arikan kernel $F_{2^m} = B_{2,m}\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}^{\otimes m}$.

Lemma 1. *Consider the m-iterated Arikan kernel $F_{2^m}, m \geq 2$. Then for any $i < 2^m$, any $\mu < m$, and any $x \in \{2^\mu t | 0 \leq t < 2^{m-\mu}\}$ one has*

$$p_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)}) = p_{x+2^\mu,x+2^{\mu+1}}(\overline{\mathcal{C}}^{(i)}) \tag{7}$$

and

$$s_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)}) = s_{x+2^\mu,x+2^{\mu+1}}(\overline{\mathcal{C}}^{(i)}) \tag{8}$$

Proof. The case of $m = 2$ has been considered in Example 2. Assume that the lemma holds for some $m \geq 2$.

Consider the case of $\mu > 0$. Observe that $F_{2^{m+1}} = \begin{pmatrix}F_{2^m} & 0\\ F_{2^m} & F_{2^m}\end{pmatrix} P$, where P is a matrix corresponding to permutation $(0, 2^m, 1, 2^m + 1, \dots)$. For $i < 2^m$ code $p_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)})$ is generated[2] by matrix

$$G_{m+1,i,x,\mu} = \begin{pmatrix}G'\\ G''\end{pmatrix}, \tag{9}$$

where G' is a $(2^m - i) \times 2^\mu$ matrix obtained by interleaving the columns of the matrix $G_{m,i,x/2,\mu-1}$ and zero columns, while G'' is a obtained by two-times repetition of columns of F_{2^m}. Since (7) is assumed to hold for F_{2^m}, it also holds for $F_{2^{m+1}}$ and $i < 2^m$. For $i \geq 2^m$ $p_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)})$ is generated by a $(2^{m+1} - i) \times 2^\mu$ matrix obtained by two-times repetition of columns of $G_{m,i-2^m,x/2,\mu-1}$. Hence, (7) again follows from the inductive assumption.

Equation (8) can be proved in the same way. □

Lemma 2. *Consider the m-iterated Arikan kernel, $m \geq 2$. Then for $x \in \{2^\mu t | 0 \leq t < 2^{m-\mu}\}$ one has $p_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)}) = p_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i-1)})$ and $s_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)}) = s_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i-1)})$, for any $i = 2^s i' > 0$, and $1 \leq \mu < m - s$.*

Proof. The case of $m = 2$ has been considered in Example 2.

Let us assume that the theorem holds for some $m \geq 2$, and consider the case of $F_{2^{m+1}}$. Due to Lemma 1, it is sufficient to prove the result for $x = 0$.

Observe that the codes $p_{0,2}(\overline{\mathcal{C}}^{(i)})$ and $s_{0,2}(\overline{\mathcal{C}}^{(i)})$, are $(2, 2, 1)$ and $(2, 1, 2)$ ones, respectively, for $0 \leq i < 2^m$, and $(2, 1, 2)$ and $(2, 0, \infty)$ ones for $2^m \leq i < 2^{m+1}$.

Consider the case of $1 < \mu < m + 1 - s$. Then for some $i = 2^s i' < 2^m$, one has $p_{0,2^\mu}(\overline{\mathcal{C}}^{(i)})$ generated by $G_{m+1,i,0,\mu}$, where G' in (9) is obtained from $G_{m,i,0,\mu-1}$. Observe that $G_{m+1,i-1,0,\mu}$ has the same structure, except that G' is obtained from $G_{m,i-1,0,\mu-1}$. Then, by inductive assumption, $G_{m,i-1,0,\mu-1}$ and $G_{m,i,0,\mu-1}$ generate the same code for $\mu - 1 < m - s$. For the case of $i = 2^s i' > 2^m$, codes $p_{0,2^\mu}(\overline{\mathcal{C}}^{(i)})$ and $p_{0,2^\mu}(\overline{\mathcal{C}}^{(i-1)})$ are generated by matrices obtained by two-times repetition of the columns of $G_{m,i-2^m,0,\mu-1}$ and $G_{m,i-2^m-1,0,\mu-1}$, respectively. Again, the inductive assumption implies that these matrices generate the same code for $\mu - 1 < m - s$.

The equality $s_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i)}) = s_{x,x+2^\mu}(\overline{\mathcal{C}}^{(i-1)})$ can be proved in the same way. □

Lemma 3. *Consider the m-iterated Arikan kernel, $m \geq 2$. Assume that for all codes $\overline{\mathcal{C}}^{(i)}$ binary sectionalization is recursively used, i.e. for any $x, y < 2^m$ the trellises are partitioned at time $z = \frac{x+y}{2}$. Then for any $i < 2^m$ the complexity of each operation (5) at recursion depth $s \leq m$ is proportional to 2^t, where $t = \begin{cases}1, & \lfloor i/2^s \rfloor \text{ even}\\ 0, & \lfloor i/2^s \rfloor \text{ odd.}\end{cases}$*

Proof. We need to estimate the cardinality of $S_z(D_y) = \left\{D_z \in p_{x,z}(\overline{\mathcal{C}}^{(i)})/s_{x,z}(\overline{\mathcal{C}}^{(i)}) | D_z \subseteq p_{x,z}(D_y)\right\}$ for $z = (x + y)/2$. Due to Lemma 1, it is sufficient to do this for the case of $x = 0, y = 2^\mu, \mu = m - s$.

The case of $m = 2$ has been studied in Example 2. Assume that the statement holds for some $m \geq 2$, and consider $F_{2^{m+1}}$.

We show that the dimension of $s_{0,2^\mu}(\overline{\mathcal{C}}^{(i)})$ is $k_{\mu,i} = 2^\mu - 1 - \lfloor i 2^{\mu-m} \rfloor$. This is obviously true for $\mu = m$. This also holds for $i = 0$ and $i = 2^m - 1$ for any $\mu \leq m$. It can be seen that $0 \leq k_{\mu,i} - k_{\mu,i-1} \leq 1$. Furthermore, $k_{\mu+1,i} \geq 2k_{\mu,i}$. According to Lemma 2, $k_{\mu,2i} = k_{\mu,2i+1}, \mu < m$. This implies that $k_{\mu,i} = \lfloor k_{\mu+1,i}/2 \rfloor$.

Hence, from (6) one obtains

$$|S_{2^{\mu-1}}(D_{2^\mu})| = 2^{k_{\mu,i} - 2k_{\mu-1,i}} = 2^{1+2\lfloor i 2^{-s-1} \rfloor - \lfloor i 2^{-s} \rfloor} = 2^t$$

□

Theorem 1. *The complexity of decoding of a polar code of length $n = 2^m$ with Arikan kernel (equivalently, the complexity of processing the m-iterated Arikan kernel) using the recursive trellis algorithm is $O(n \log n)$*

Proof. Lemma 1 implies that exactly the same operations are performed at all branches of the recursion tree on any fixed level. Lemma 2 implies that the recursion depth is identical for all branches. In particular, there are 2^{m-s} indices i, where depth-s recursion is used. Hence, the total number of operations (5) performed at each node of the recursion tree is $\sum_{s=0}^{m} 2^{m-s} \cdot 2^s = n \log_2 n$. Lemma 3 implies that the complexity of each such operation is $O(1)$. □

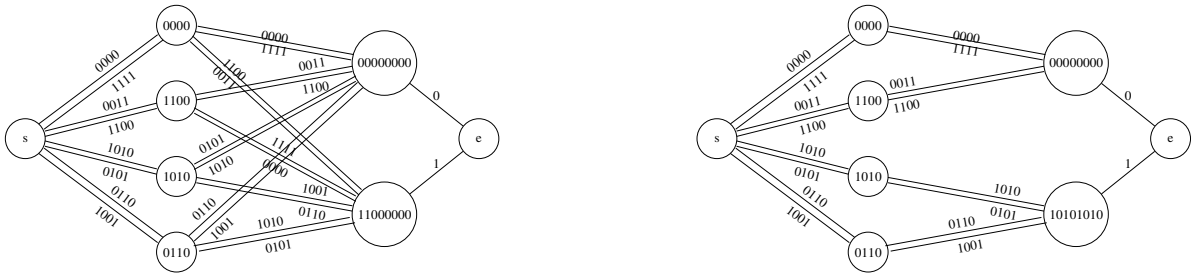
B. Sorted Arikan kernel

It was shown in [15] that the sorted Arikan kernel[3] of dimension $l = 8$

$$F_8 = \begin{pmatrix}1&0&0&0&0&0&0&0\\ 1&0&0&0&1&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&0&1&0&1&0&1&0\\ 1&1&0&0&1&1&0&0\\ 1&1&1&1&0&0&0&0\\ 1&1&1&1&1&1&1&1\end{pmatrix}$$

[2] This matrix may have some linearly dependent rows.

[3] The columns of the original kernel were permuted to minimize the complexity of the recursive processing algorithm.



(a) $\overline{\mathcal{C}}^{(3)}$ (b) $\overline{\mathcal{C}}^{(4)}$

Fig. 2: Trellises for the sorted Arikan kernel

has scaling exponent 3.577, which is better compared to the Arikan kernel. The proposed approach can be immediately used to implement processing of such kernel. Processing of phases 0–2 and 5–7 is identical to that of 3-iterated Arikan kernel.

The coset representatives of $p_{0,8}(\overline{\mathcal{C}}^{(3)})/s_{0,8}(\overline{\mathcal{C}}^{(3)})$ are $D'_8 = (0,0,0,0,0,0,0,0)$ and $D''_8 = (1,1,0,0,0,0,0,0)$. It can be seen that $p_{0,4}(\overline{\mathcal{C}}^{(3)}) = p_{5,8}(\overline{\mathcal{C}}^{(3)})$ is a $(4,3,2)$ code, while $s_{0,4}(\overline{\mathcal{C}}^{(3)}) = s_{5,8}(\overline{\mathcal{C}}^{(3)})$ is a $(4,1,4)$ code. Hence, the states of a two-section trellis at time 4 are given by $S_4(D'_8) = S_4(D''_8) = \{(0,0,0,0),(1,0,1,0),(1,1,0,0),(0,1,1,0)\}$. It can be also verified that $s_{x,x+2}(\overline{\mathcal{C}}^{(3)}), x \in \{0,2,4,6\}$ is the trivial $(2,0,\infty)$ code, while $p_{x,x+2}(\overline{\mathcal{C}}^{(3)})$ is the $(2,2,1)$ code.

Similarly, the coset representatives of $p_{0,8}(\overline{\mathcal{C}}^{(4)})/s_{0,8}(\overline{\mathcal{C}}^{(4)})$ are $\widetilde{D}'_8 = (0,0,0,0,0,0,0,0)$ and $\widetilde{D}''_8 = (1,0,1,0,1,0,1,1)$. It can be also seen that $p_{0,4}(\overline{\mathcal{C}}^{(4)}) = p_{5,8}(\overline{\mathcal{C}}^{(4)}) = p_{0,4}(\overline{\mathcal{C}}^{(3)}) = p_{5,8}(\overline{\mathcal{C}}^{(3)})$ and $s_{0,4}(\overline{\mathcal{C}}^{(4)}) = s_{5,8}(\overline{\mathcal{C}}^{(4)}) = s_{0,4}(\overline{\mathcal{C}}^{(3)}) = s_{5,8}(\overline{\mathcal{C}}^{(3)})$. This implies that $S_4(\widetilde{D}'_8) = \{(0,0,0,0),(1,1,0,0)\}$ and $S_4(\widetilde{D}''_8) = \{(1,0,1,0),(0,1,1,0)\}$. Figure 2 shows the corresponding trellises.

The same approach can be used to obtain trellis-based processing algorithms for larger kernels. A naive implementation of this method would require more arithmetic operations compared to highly optimized techniques like [9]. However, the complexity can be substantially reduced by employing LLR-domain computations. We do not describe here these tricks due to lack of space.

V. Conclusions

It was shown that the recursive trellis decoding method can be used to implement processing of large polarization kernels. The processing complexity can be substantially reduced by reusing the intermediate results obtained at different phases. Furthermore, the Arikan SC decoding algorithm appears to be an an instance of this approach.

Due to vast literature available on trellises, this result may lead to construction of large polarization kernels of arbitrary dimension, which admit efficient processing, as well as to improve understanding of the corresponding polar codes. In particular, for a fixed polarization rate, one may look for kernels, which induce section codes $p_{x,y}(\overline{\mathcal{C}}^{(i)}), s_{x,y}(\overline{\mathcal{C}}^{(i)})$ with small $|p_{x,y}(\overline{\mathcal{C}}^{(i)})/s_{x,y}(\overline{\mathcal{C}}^{(i)})|$, and keep these section codes invariant for as many phases i as possible. One may also look for section codes, which admit reduced complexity decoding. The construction of polar codes may be also tweaked by replacing some section codes, to obtain improved performance.

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. H. Hassani, K. Alishahi, and R. Urbanke, "Finite-length scaling for polar codes," *IEEE Transactions On Information Theory*, vol. 60, no. 10, October 2014.

[3] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions On Information Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.

[4] P. Trifonov and V. Miloslavskaya, "Polar subcodes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 254–266, February 2016.

[5] M. Mondelli, S. H. Hassani, and R. Urbanke, "Scaling exponent of list decoders with applications to polar codes," *IEEE Transactions On Information Theory*, vol. 61, no. 9, September 2015.

[6] 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, December 2010.

[7] A. Fazeli, S. H. Hassani, M. Mondelli, and A. Vardy, "Binary linear codes with optimal scaling: Polar codes with large kernels," in *Proceedings of IEEE Information Theory Workshop*, 2018.

[8] V. Bioglio and I. Land, "On the marginalization of polarizing kernels," in *Proceedings of International Symposium on Turbo Codes and Iterative Information Processing*, 2018.

[9] G. Trofimiuk and P. Trifonov, "Efficient decoding of polar codes with some 16×16 kernels," in *Proceedings of IEEE Information Theory Workshop*, 2018.

[10] 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.

[11] T. Fujiwara, H. Yamamoto, T. Kasami, and S. Lin, "A trellis-based recursive maximum-likelihood decoding algorithm for binary linear block codes," *IEEE Transactions On Information Theory*, vol. 44, no. 2, March 1998.

[12] E. Blokh and V. Zyablov, "Coding of generalized concatenated codes," *Problems of Information Transmission*, vol. 10, no. 3, pp. 45–50, 1974.

[13] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3221 – 3227, November 2012.

[14] H. Griesser and V. R. Sidorenko, "A posteriory probability decoding of nonsystematically encoded block codes," *Problems of Information Transmission*, vol. 38, no. 3, 2002.

[15] A. Fazeli and A. Vardy, "On the scaling exponent of binary polarization kernels," in *Proceedings of 52nd Annual Allerton Conference on Communication, Control and Computing*, 2014, pp. 797 – 804.