Recursive Trellis Processing of Large Polarization Kernels

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

Abstract—A reduced complexity algorithm is presented for computing the log-likelihood ratios arising in the successive cancellation decoder for polar codes with large kernels of arbitrary dimension. The proposed algorithm exploits recursive trellis representation of the codes generated by submatrices of the polarization kernel, and enables polar codes based on large kernels to provide better performance compared to the codes based on Arikan kernel with the same decoding complexity.

I. INTRODUCTION

Polar codes [1], have already made their way into the 5G specification. However, their role is limited there for encoding of small data blocks, since long polar codes do not compete well with e.g. LDPC codes. Polar codes with large kernels were shown to have asymptotically optimal scaling exponent [2]. The SC decoding algorithm for such codes has complexity $O(n \log n)$ operations of computing LLRs. Each such operation, called kernel processing or kernel marginalization [3], has complexity $O(2^l)$ if implemented straightforwardly, where $l$ is kernel dimension. An approximate kernel processing method based on box-and-match decoder was suggested in [4]. Some kernels of dimension 16 and 32 were published together with efficient processing algorithms [5]. The corresponding polar codes were shown to have lower decoding complexity compared to codes based on Arikan kernel with the same performance. However, no efficient processing techniques for generic kernels have been published so far.

In this paper we present such algorithm, and report its complexity for some kernels available in the literature. We show that the proposed approach has much lower complexity compared to the Viterbi algorithm.

II. BACKGROUND

A. Polar codes

Polar code [6] is a set of vectors $u_0^{n-1} = u_0^{n-1} K^\otimes m$, where $K$ is a non-singular $l \times l$ matrix called polarization kernel, $n = l^m$, $u_i = 0$ for $i \in \mathcal{F}$, $\mathcal{F} \subset [n]$ is a frozen set, and $[n] = \{0, \ldots, n-1\}$. This definition can be generalized to obtain mixed kernel polar codes with the codewords given by $c_0^{n-1} = u_0^{n-1} (K_{t_1} \otimes K_{t_2} \otimes \cdots \otimes K_{t_m})$ [7], [8], where $K_{t_i}$ is a kernel of dimension $l_i$. Unless stated otherwise, we consider here the case of all kernels being the same.

Decoding of polar codes can be implemented by the successive cancellation algorithm, which makes decisions

$$\hat{u}_i = \begin{cases} 0, & i \in \mathcal{F} \\ \arg \max_{u_i \in \mathcal{F}_2} W_m^{(i)}(s^{i-1} \cdot u_i | r_0^{n-1}), & i \notin \mathcal{F}, \end{cases}$$

where $W_m^{(i)}(u_0 | r_0^{n-1}) = \sum_{u_i+1} \prod_{j=0}^{n-1} W_0^{(i)}((u_0^{n-1} K^\otimes l_i) | r_i)$, $W_0^{(i)}(c | r) = W_0^{(i)}(c | r) = \frac{W(r)}{2^b(r)}$, $W(r)$ is the channel transition probability, and $\cdot$ denotes the concatenation operator. These probabilities can be recursively computed as

$$W_m^{(l+s)}(u_0 | r_0^{n-1}) = \sum_{u_i+1} \prod_{j=0}^{l-1} W_m^{(j)}((u_0^{l+1} - 1) K)_{j,t} \in [i+1],[r_0^{n-1}],$$

where $r_0^{n-1} = (r_j, r_{j+1}, \ldots, r_{j+n-1})$. This operation is known as kernel processing or kernel marginalization [9], [3].

It was suggested in [4] to approximate the above probabilities as

$$W_m^{(i)}(u_0 | r_0^{n-1}) \approx W_m^{(i)}(u_0 | r_0^{n-1}) = \max_{u_i+1} \prod_{j=0}^{l-1} W_0^{(i)}((u_0^{l} - 1) K)_{j,r_0^{n-1}},$$

so that

$$W_m^{(l+s)}(u_0 | r_0^{n-1}) = \max_{u_i+1} \prod_{j=0}^{l-1} W_m^{(j)}((u_0^{l+1} - 1) K)_{j,t} \in [i+1],[r_0^{n-1}].$$

Decoding can be implemented in the LLR domain using LLRs

$$S_m^{(i)}(u_0 | r_0^{n-1}) = \ln W_m^{(i)}(u_0^{i-1} \cdot 0 | r_0^{n-1}).$$

Assume for the sake of simplicity $m = 1$. It can be seen that

$$S_1^{(i)}(u_0^{i-1} | r_0^{n-1}) = \max_{u_i+1} \prod_{j=0}^{l-1} \ln W_0^{(i)}((u_0^{i-1} \cdot 0 \cdot u_i+1) K)_{j,r_0^{n-1}},$$

$$- \max_{u_i+1} \prod_{j=0}^{l-1} \ln W_0^{(i)}((u_0^{i-1} \cdot 1 \cdot u_i+1) K)_{j,r_0^{n-1}},$$

$$= \frac{1}{2} \max_{u_i+1} Q((u_0^{i-1} \cdot 0 \cdot u_i+1) K, r_0^{l-1}),$$

$$- \frac{1}{2} \max_{u_i+1} Q((u_0^{i-1} \cdot 1 \cdot u_i+1) K, r_0^{l-1}),$$

where $Q(l_0^{l-1}, l_0^{l-1}) = \sum_{j=0}^{l-1} S_0^{(0)}(r_j)$ is the correlation function. The objective of the present paper is to provide an efficient method for computing LLRs $S_1^{(i)}(u_0^{i-1} | r_0^{n-1})$.

B. Recursive maximum likelihood decoding algorithm

It can be seen that computing (3) reduces to ML decoding in the cosets of the code generated by $l - i - 1$ last rows of
matrix $K$, provided that all symbols $s_x, s \in [l]$, are equiprobable binary random values. An efficient recursive maximum likelihood decoding algorithm for linear block codes was suggested in [10]. The idea is to recursively partition the received noisy vector into a number of sections $[x, y]$, identify for each section a number of most likely vectors $v_x^{y-1} \in \mathbb{F}_2^y$ corresponding to the received values $r_x^y$, and combine them to obtain most likely vectors for longer sections.

Given a linear code $C$, let $C_{h, h'}$ be its subcode, such that all its codewords have non-zero symbols only in positions $h \leq i < h'$. Let $p_{h, h'}(C)$ be a linear code obtained by puncturing all symbols, except those in positions $h \leq i < h'$, 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 $h \leq i < h'$. The codes $s_{h, h'}(C)$ and $p_{h, h'}(C)$ are 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 $p_{x, y}(C)/s_{x, y}(C)$ [10]. This coset may appear in the trellis several times. Hence, one can simplify maximum likelihood decoding (ML) decoding 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 correlation $Q(D) = Q(l(D), r_y^{y-1})$. Let the composite branch table (CBT) $T_{x,y}$ be an array containing values $T_{x,y}[v] = l(D)$ and $T_{x,y}[v].q = Q(D)$, where $v$ is an index of $D$. In the case of conventional decoding of an $(n, k)$ code, $p_{0,n}(C)/s_{0,n}(C)$ contains a single element, so the corresponding CBT has one entry $T_{0,n}[0]$, which gives 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 $p_{x, y}(C)/s_{x, y}(C)$. We assume that this method is used for $y-x \geq 2$. However, more efficient approach was suggested in [10] for the case of $y-x \geq 2$. Consider some $z : x < z < y$. Let the generator matrix of $p_{x, y}(C)$ be represented as

$$G^{(p)}_{x,y} = \begin{pmatrix} G^{(s)}_{x,y} & 0 \\ 0 & G^{(0)}_{x,y} \\ G^{(10)}_{x,y} & G^{(11)}_{x,y} \end{pmatrix}, \quad (4)$$

where $G^{(s)}_{x,y} = \begin{pmatrix} G^{(s)}_{x,z} & 0 \\ 0 & G^{(s)}_{z,y} \end{pmatrix}$ is a generator matrix of $s_{x,y}(C)$, and $G^{(0)}_{x,y}, G^{(0)}_{y,x}$ are some $k_x^y \times (y-z)$ and $k_y^x \times (y-z)$ matrices, respectively, where $k_x^y = k_y^x(C)$ and $k_y^x = k_y^x(C)$ are some code-dependent integers. There is a one-to-one correspondence between vectors $vG^{(s)}_{x,y}$, where $G^{(s)}_{x,y} = \begin{pmatrix} G^{(s)}_{x,y} \\ G^{(0)}_{x,y} \end{pmatrix}$ is a $k_x^y \times (y-x)$ matrix, and cosets $D \in p_{x, y}(C)/s_{x, y}(C)$. Here $k_x^y, k_y^x$ are some integers, which depend on code structure, and can be obtained from the minimum span form of its generator matrix.

Hence, we write $T_{x,y}[v], l := l(D)$ and $T_{x,y}[v].q = Q(D)$, with $D$ being a coset corresponding to $v$. It can be seen that

$$T_{x,y}[v].q = \max_{c^{y-1} \in D} Q(c_x^{y-1}, r_y^{y-1}) = \max_{w \in \mathbb{F}_2^{x-y}} \left( T_{x,z}[w].q + T_{z,y}[b].q \right), v \in \mathbb{F}_2^y \quad (5)$$

where $a$ and $b$ are indices of the cosets $D' \in p_{x, z}(C)/s_{x, z}(C)$ and $D'' \in p_{z, y}(C)/s_{z, y}(C)$, respectively, such that

$$(w, v) \begin{pmatrix} G^{(0)}_{x,z} \\ G^{(10)}_{x,z} \end{pmatrix} \in D' \quad \text{and} \quad (w, v) \begin{pmatrix} G^{(0)}_{z,y} \\ G^{(11)}_{z,y} \end{pmatrix} \in D''.$$  

Such vectors $a, b$ can be identified from the system of equations

$$\begin{cases} (a') \quad (G^{(s)}_{x,z} \quad G^{(s)}_{z,y}) = (w \quad v) \begin{pmatrix} G^{(0)}_{x,y} \\ G^{(10)}_{x,y} \end{pmatrix}, \\ (b') \quad (G^{(s)}_{x,y} \quad G^{(s)}_{y,z}) = (w \quad v) \begin{pmatrix} G^{(0)}_{x,y} \\ G^{(11)}_{x,y} \end{pmatrix}, \end{cases}$$

where $a', b'$ are not used elsewhere. The solution is given by $a = (w \quad v)G^{(s)}_{x,y}$ and $b = (w \quad v)G^{(s)}_{y,z}$ for some matrices $G^{(s)}_{x,y}$ and $G^{(s)}_{y,z}$. The corresponding most likely coset representative is given by $T_{x,y}[v].l = T_{x,z}[a], l \cdot T_{x,z}[b], l$, where $a, b$ are the values of $a$ and $b$, which deliver maximum in (5).

The complexity of this calculation is $O(K_x^y \cdot K_z^y)$, It can be further reduced by exploiting the tricks suggested in [10]. The overall decoding complexity strongly depends on the sectionalization method being used, i.e. a rule for selection of the partitioning point $z$ for some $x, y$. This approach, known as recursive maximum likelihood decoding (RMLD), was shown to be more efficient compared to the Viterbi algorithm [10].

III. RECURSIVE TRELlIS PROCESSING

A. Extended kernel codes

Polar codes can be considered as a result of recursive application of the construction of generalized concatenated codes [11], [12]. These codes rely on non-systematic inner codes. An optimal SISO decoding algorithm for non-systematically encoded linear block codes was presented in [13]. This algorithm can be easily tailored to implement computation of (3). To do this, consider an extended $(l+1, l-i)$ code $\overline{C}^{(i)}$ generated by matrix $G^{(i)}$. First $l$ columns of this matrix are obtained by taking $l - i$ last rows of kernel $K$. The last column has 1 in the 0-th row, and zeroes in the remaining positions. Assuming that $u_0^{l-1} = 0$, one obtains that computing (3) is equivalent to finding the most probable codewords of code $\overline{C}^{(i)}$ having 0 and 1 in the last symbol. This can be implemented by running the Viterbi algorithm over the trellis of $\overline{C}^{(i)}$, assuming that the last codeword symbol is erased. The same trellises, although with different labeling, can be used to implement decoding in the cosets of $\overline{C}^{(i)}$, which arise in the case of $u_0^{l-1} \neq 0$.

Example 1. Consider 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 $G^{(1)} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$.
B. Recursive processing of polarization kernels

We propose to compute (3) by applying the RMLD algorithm to the cosets of the extended kernel codes. Let \( \overline{C}(u_0^{-1}) = w + \overline{C}(i) \) be the coset of \( C(i) \) given by prior decisions \( u_0^{-1} \), where \( w = (u_0^{-1}K_0, ..., u_0^{-1}, 0) \), and \( K_0, ..., u_0^{-1} \) is the matrix consisting of \( i \) top rows of \( K \). For any section \((x,y)\), the cosets associated with states in the recursive trellis for \( \overline{C}(u_0^{-1}) \) are obtained from those for \( \overline{C}(i) \) as \( D(u_0^{-1}) = \{ f + u_0^{-1}|f \in D \} \), \( D \in p_{xy}(\overline{C}(i))/s_{xy}(\overline{C}(i)) \).

It can be seen that \( p_{0,i}(\overline{C}(i))/s_{0,i}(\overline{C}(i)) \) contains two cosets, which correspond to \( u_1 = 0 \) and \( u_1 = 1 \). Hence,
\[
S_{1}(i)(u_0^{-1}, r_0^{-1}) = \frac{T_{0,i}[0], q - T_{0,i}[1], q}{2},
\]
where \( T_{0,i} \) is the composite branch table constructed for \( \overline{C}(u_0^{-1}) \) given noisy vector \( r_0^{-1} \). It is assumed in what follows that CBT entries contain only \( q \) values, and the corresponding coset representatives \( l(D) \) are omitted.

Furthermore, we propose to reuse the composite branch tables, or their parts, obtained at successive phases \( i \). To do this, we need to identify how CBTs evolve with \( i \), find a way to handle prior decisions \( u_0^{-1} \), design efficient algorithms for construction of CBTs for short sections, and obtain an optimal sectionalization strategy.

1) Reusing the CBTs: Assume that the same sectionalization is used for all phases \( i \). Obviously, \( p_{xy}(\overline{C}(i)) = p_{xy}(\overline{C}(i)) \) and \( s_{xy}(\overline{C}(i)) \) hold for any \((x,y)\), such that \( 0 \leq x < y \leq 1 \).

**Lemma 1.** If \( p_{xy}(\overline{C}(i)) = p_{xy}(\overline{C}(i)) \) and \( s_{xy}(\overline{C}(i)) = s_{xy}(\overline{C}(i)) \), then \( u_{i,j} \in \mathbb{F}_2 \) the composite branch table \( T_{x,y} \) constructed for \( \overline{C}(u_0^{-1}) \) is identical to the one constructed for \( \overline{C}(u_0^{-1}) \), denoted \( T_{x,y} \), for the same received vector \( r_0^{-1} \).

**Proof.** Since both section codes are identical for phases \( i \) and \( i+1 \), the CBTs have the same size, \( p_{xy}(\overline{C}(i)) = p_{xy}(\overline{C}(i)) \) implies that \( \kappa = (K_{i,x}, ..., K_{i,y-1}) \in p_{xy}(\overline{C}(i)) \). Hence, for any \( u_{i,j} \in \mathbb{F}_2 \) both \( p_{xy}(\overline{C}(i))/s_{xy}(\overline{C}(i)) \) and \( p_{xy}(\overline{C}(i))/s_{xy}(\overline{C}(i)) \) have cosets containing \( u_{i,j} \), for \( u_{i,j} \in \mathbb{F}_2 \). Hence, both \( T_{x,y} \) and \( T_{x,y} \) contain the same values.

The above lemma suggests that one does not need to recompute the CBTs for those sections, where section codes do not change from phase \( i \) to phase \( i+1 \).

In what follows, we assume that \( s_{x,z}(C(i)) = s_{x,z}(C(i)) \) and \( s_{y,z}(C(i)) = s_{y,z}(C(i)) \). (7)

Let us assume temporarily that \( u_0^{-1} = 0 \).

Even if section codes do change, it is still possible to reuse some results obtained at prior phases. Let \( k_{i,x,y} = k_{i,x,y}^{(i)} \) and \( k_{i,x,y}^{(i)} = k_{i,x,y}^{(i)} \). First, observe that if \( k_{i+1,x,y}^{(i)} = k_{i+1,x,y}^{(i)} \), then \( p_{xy}(C(i))/s_{xy}(C(i)) \) is not a coset representative for \( \overline{C}(u_0^{-1}) \).

Second, we propose to implement maximization recursively, and keep all intermediate results. More specifically, we propose to rewrite (5) as
\[
T_{x,y}[v] = \max_{w \in F_{2}^{k_{i,x,y}^{(i)}}} (T_{x,y}[w] + T_{x,y}[v])
= \max_{w \in F_{2}^{k_{i,x,y}^{(i)}}} \cdots \max (T_{x,y}[w] + T_{x,y}[v]).
\]

Instead of storing the CBT the final results of maximization in (8), we propose to keep the intermediate results of maximization for all \( w \). These values can be arranged in a binary tree for each \( v \in F_{2}^{k_{i,x,y}^{(i)}} \), so that a path from a root in this tree can be specified by values \( w_{k_{i,x,y}^{(i)}-1}, w_{k_{i,x,y}^{(i)}-2}, ..., w_{k_{i,x,y}^{(i)}} \). By maximization forest we denote the set of such trees obtained at some phase for a given section. The subtrees within a forest can be indexed by variables \( w, v \).

We propose to use the above described maximization tree constructed at some phase \( i_0 \) to obtain CBTs for all \( i \geq i_0 \), where (7) holds. Let \( i_1 > i_0 \) be the smallest integer, where this does not hold.

**Lemma 2.** Let \( G_{i,x,y} = (G_{i,x,y}^{(0)} G_{i,x,y}^{(1)}) \) and \( G_{i,x,y}^{(0)} = (G_{i,x,y}^{(0)} G_{i,x,y}^{(1)}) \) be the matrices obtained from (4) for code \( C(i) \) for any \( i \). If all matrices \( G_{i,x,y}^{(0)} \) are nested, so that \( G_{i+1,x,y}^{(0)} \) occupies top rows of \( G_{i,x,y}^{(0)} \) for any \( i : i_0 \leq i \leq i_1 \), then the maximization forest for phase \( i : i_0 \leq i \leq i_1 \) can be obtained by taking the subtrees of the trees in the forest constructed at phase \( i_0 \), given by values \( w \) and \( v \) satisfying the equation
\[
(w, v) = M_{i,x,y}(w, v) = \left( \begin{array}{c} w \\ v \\ \end{array} \right) = \left( \begin{array}{c} G_{i,x,y}^{(0)} G_{i,x,y}^{(1)} \\ M_{i,x,y} \\ \end{array} \right),
\]
where \( w, v \in F_{2}^{k_{i,x,y}^{(0)}}, v \in F_{2}^{k_{i,x,y}^{(1)}} \) denote the subtree indices in the forest at phase \( i \).

**Proof.** Assumption (7) ensures that the maximization trees for all \( i \) are obtained from the same values.

Observe that the matrices in (9) together with the generator matrices of \( s_{x,z}(C(i)) \) and \( s_{y,z}(C(i)) \) constitute generator matrices of \( p_{xy}(C(i)) \) and \( p_{xy}(C(i)) \), given by (4). Since \( p_{xy}(C(i)) \subset p_{xy}(C(i)) \), for any \( w, v \) there is a unique solution \( w, v = M_{i,x,y} \).

Hence, for each \( \overline{w} \) the corresponding maximization tree indeed appears as a subtree indexed by \( w_j \) of the \( v \)-th maximization tree.

It is always possible to obtain \( G_{i,x,y}^{(0)} \) in the form required by Lemma 2 by applying elementary row operations.

2) Handling prior decisions: The SC decoder needs to take into account at phase \( i \) the values \( u_0^{-1} \). This reduces to decoding in cosets of section codes. The corresponding coset representative for section \((x,y)\) can be computed as a linear combination of subvectors \((K_{j,x} K_{j,x+1} \cdots K_{j,y-1})\) by \( j \leq i \), of the kernel.
Consider first the case of section \([x, y]\), where the maximization forest \((8)\) is constructed from scratch. If there exists a solution of

\[
(f_{j,x,y} \ h_{j,x,y}) \ (G^{(s)}_{j,x,y} G^{(s)}_{j,x,y}) = (K_{j,x} \ K_{j,x+1} \ldots \ K_{j,y-1}),
\]

then the CBT for section \([x, y]\) contains at position \(h_{j,x,y}\) the required coset representative. In this case we assume \(h_{j,x,y} = 0\). Given a vector of prior decisions \(u_0^{i-1}\), we obtain the corresponding position offset at section \([x, y]\) as \(h_{x,y} = \sum_{j=0}^{i-1} u_j h_{j,x,y}\), so that \((8)\) becomes

\[
T_{x,y}[v], q = \max \left( T_{x,z}[a + h_{x,z}], q + T_{z,y}[b + h_{z,y}], q \right). \tag{11}
\]

The values \(h_{x,y}\) are similar to partial sums, which arise in the SC decoder of Arikan polar codes.

For those sections, where the CBT is obtained by taking subtrees of \((8)\), we need to check if there is a solution of

\[
\omega_{j,x,y} \left( G^{(s)}_{0o,x,y} G^{(s)}_{0o,x,y} \right) = (K_{j,x} \ K_{j,x+1} \ldots \ K_{j,y-1}). \tag{12}
\]

If this equation does not have a solution for some \(j < i\), the corresponding coset representatives have already been accounted for at smaller sections while constructing the CBT for section \([x, y]\) at phase \(i_0\) and we assume \(\omega_{j,x,y} = 0\). Otherwise, the corresponding CBT can be obtained from the maximization forest constructed at phase \(i_0 < i\) by taking entries

\[
(w \ v) = (w \ \pi) M_{i,x,y} + \sum_{j=0}^{i-1} u_j \omega_{j,x,y}.
\]

3) Special trellises: It can be seen that the result of \((6)\) does not change if the same value is subtracted from all CBT entries at any section. Doing this enables one to compute some CBTs with smaller number of operations compared to \((11)\). The following special cases were identified:

1) \(k_{l,x,y} = k_{l,x,y} = 1, \hat{G}_{i,x,y} = \left( \begin{array}{c} 0 \\ 1 \end{array} \right), \ G_{i,x,y} = \left( \begin{array}{c} 1 \\ 1 \end{array} \right)\).

Let \(a = T_{x,z}[0] q = T_{x,y}[1] q, b = T_{x,z}[0] q - T_{x,y}[1] q\). We propose to set \(T_{x,z}[0], q = \text{sgn}(a) \text{sgn}(b) \min(1, |a|, |b|)\) and \(T_{x,y}[1], q = -T_{x,z}[0], q\). If one can guarantee that \(T_{x,z}[1], q = -T_{x,z}[0], q\), then it is possible to further simplify computation by setting \(a = T_{x,z}[0], q\). Similar simplification applies to \(b\). This trick allows one to construct section CBT using just 1 comparison operation, instead of 2 comparison and 4 summations for a straightforward implementation.

2) \(k_{l,x,y} = 1, k_{l,x,y} = 0, \hat{G}_{i,x,y} = \hat{G}_{i,x,y} = \left( \begin{array}{c} 1 \\ 0 \end{array} \right), \ G_{i,x,y} = \left( \begin{array}{c} 0 \\ 1 \end{array} \right)\).

Using the same definitions as above, we propose to set \(T_{x,y}[0], q = a + b, T_{x,y}[1], q = b - a, T_{x,y}[2], q = -T_{x,y}[1], q, T_{x,y}[3], q = -a\).

4) If there is all-1 row in \(G_{i,x,y}\), then one step of maximization can be avoided in \((8)\) by taking the absolute values of the corresponding terms. This reduces the maximization tree construction complexity by a factor of 2.

Simplification tricks 1 and 2 together with the results in [14] fully establish the equivalence of the proposed approach and the min-sum SC algorithm for the case of \(K = B_{\mu} \left( \begin{array}{c} 1 \\ 1 \\ 0 \end{array} \right)\), where \(B_{\mu}\) is the bit reversal permutation matrix.

4) Optimal sectionalization: The total complexity of kernel processing with the proposed algorithm is given by

\[
C = \sum_{i=0}^{l-1} (\delta_i + c_{i,0,0}),
\]

where \(\delta_i\) is the complexity of computing the final LLR from the obtained CBT, and \(c_{i,x,y}\) is the complexity of construction of the CBT for section \([x, y]\) at phase \(i\). In general, the former operation reduces to computing \((6)\), i.e. costs 1 subtraction. However, if above described special trellises 1 or 2 arise at section \([0, l]\), then \(\delta_i = 0\). Furthermore,

\[
c_{i,x,y} = \begin{cases} m_{i,x,y}, & \text{if CBTs for subsections can be reused} \\ m_{i,x,y} + c_{i,x,z} + c_{i,z,y}, & \text{otherwise,} \end{cases}
\]

where

\[
m_{i,x,y} = \begin{cases} M_{2j}, & \text{if type-}j \text{ special trellis is encountered,} \\ 2k_{i,x,y} + k_{i,x,y} - f_{i,x,y} + 2k_{i,x,y} (2k_{i,x,y} - f_{i,x,y} - 1), & \text{otherwise,} \end{cases}
\]

\(f_{i,x,y}\) is equal 1 if the above described 4-th simplification trick is applicable, and 0 otherwise, and \(M_{2j}\) is the complexity of type-\(j\) special trellis, \(1 \leq j \leq 3\). The first and second terms in the latter expression represent the number of summations and comparisons, respectively. For each section \([x, y]\) one should find splitting position \(z\), which minimizes the overall complexity.

This can be done offline by the optimization algorithm given in [10]. Observe that sectionalization should be optimized jointly for all phases \(i\).

IV. NUMERIC RESULTS

Table I presents processing complexity for some polarization kernels. Here \(K_{l}\) denotes a kernel of size \(l\). For comparison, we present the complexity of the window-based and Viterbi processing algorithms. To the best of our knowledge, no other processing algorithm was published for these kernels. It can be seen that for kernels given in [15] the proposed approach has slightly higher complexity compared to the window processing algorithm. However, extension of the latter algorithm to kernels of size other than \(2^\mu\) is non-obvious, while the recursive trellis algorithm can be applied to any
TABLE I: Kernel processing complexity

<table>
<thead>
<tr>
<th>Kernel $K_t$</th>
<th>$E(K_t)$</th>
<th>$\mu(K_t)$</th>
<th>$\nu(K_t)$</th>
<th>State of the art</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{24}B_4$</td>
<td>0.51828</td>
<td>3.45</td>
<td></td>
<td>window 95</td>
<td>131</td>
</tr>
<tr>
<td>$K_{32}B_4$</td>
<td>0.621936</td>
<td>3.417</td>
<td></td>
<td>window 297</td>
<td>406</td>
</tr>
<tr>
<td>$K_{24}$</td>
<td>0.52194</td>
<td>3.4211</td>
<td></td>
<td>Viterbi</td>
<td>355</td>
</tr>
<tr>
<td>$K_{32}$</td>
<td>0.53606</td>
<td>3.1221</td>
<td></td>
<td>Viterbi 99745</td>
<td>28455</td>
</tr>
</tbody>
</table>

Note: The proposed method involves matrix manipulations, which should be performed once at the design time. Actual kernel processing reduces to summation and comparison of the elements of some submatrices of the polarization kernel. The proposed algorithm can be applied to kernels of arbitrary dimension, and has much lower complexity compared to the Viterbi algorithm. Derivation of a processor for a given kernel according to the proposed method involves matrix manipulations, which should be performed once at the design time. Actual kernel processing reduces to summation and comparison of the elements of some arrays, where the indices of the operands are obtained as XOR of some pre-computed values and partial sums given by the SC algorithm decisions. The obtained algorithm is fully deterministic.

The proposed algorithm enables polar subcodes with well-designed large kernels to provide better performance/complexity tradeoff compared to the codes based on Arikan kernel as well as LDPC codes. However, finding large kernels with good polarization properties and low processing complexity remains, in general, an open problem.

Fig. 1: Performance and SCL decoding complexity

Fig. 2: Performance and decoding complexity of long codes

In this paper a novel processing algorithm for large polarization kernels was proposed. It relies on extensive reuse of the intermediate results arising in the recursive maximum likelihood decoding algorithm for the codes generated by submatrices of the polarization kernel. The proposed algorithm can be applied to kernels of arbitrary dimension, and has much lower complexity compared to the Viterbi algorithm. Derivation of a processor for a given kernel according to the proposed method involves matrix manipulations, which should be performed once at the design time. Actual kernel processing reduces to summation and comparison of the elements of some arrays, where the indices of the operands are obtained as XOR of some pre-computed values and partial sums given by the SC algorithm decisions. The obtained algorithm is fully deterministic.

The proposed algorithm enables polar subcodes with well-designed large kernels to provide better performance/complexity tradeoff compared to the codes based on Arikan kernel as well as LDPC codes. However, finding large kernels with good polarization properties and low processing complexity remains, in general, an open problem.

V. CONCLUSIONS

In this paper a novel processing algorithm for large polarization kernels was proposed. It relies on extensive reuse of the intermediate results arising in the recursive maximum likelihood decoding algorithm for the codes generated by submatrices of the polarization kernel. The proposed algorithm can be applied to kernels of arbitrary dimension, and has much lower complexity compared to the Viterbi algorithm. Derivation of a processor for a given kernel according to the proposed method involves matrix manipulations, which should be performed once at the design time. Actual kernel processing reduces to summation and comparison of the elements of some arrays, where the indices of the operands are obtained as XOR of some pre-computed values and partial sums given by the SC algorithm decisions. The obtained algorithm is fully deterministic.

The proposed algorithm enables polar subcodes with well-designed large kernels to provide better performance/complexity tradeoff compared to the codes based on Arikan kernel as well as LDPC codes. However, finding large kernels with good polarization properties and low processing complexity remains, in general, an open problem.

ACKNOWLEDGEMENT

This work is partially supported by the Ministry of Science and Higher Education of Russian Federation, passport of goszadanie no. 2019-0898.
REFERENCES


