# Block Sequential Decoding of Polar Codes

Grigorii Trofimiuk, Peter Trifonov
Peter the Great St. Petersburg Polytechnic University
Email: {grigoriyt,petert}@dcn.icc.spbstu.ru

***Abstract*—The problem of efficient decoding of polar codes is considered. A modification of the sequential decoding algorithm is proposed. Instead of processing one symbol at each iteration, we perform ML decoding of blocks of symbols using the tree-trellis list Viterbi algorithm. Numeric results show that the proposed approach significantly reduces the average complexity and latency of the sequential decoder without any performance degradation.**

## I. Introduction

Polar codes were recently shown to be able to achieve the capacity of a wide class of communication channels [1]. The successive cancellation (SC) algorithm fails to achieve the ML performance. This problem can be avoided by employing a list decoding algorithm [2], [3]. The sequential decoding algorithm was shown to have much lower complexity [4].

In this paper a generalization of the sequential decoding algorithm is introduced, which enables one to significantly reduce the decoding complexity and latency. The idea is to recursively decompose the polar code as a Plotkin concatenation of some outer codes. The recursion is stopped as soon as codes efficiently decodable using a ML algorithm are obtained. Instead of splitting paths in the code tree at each phase corresponding to a non-frozen symbol, as in classical list/stack algorithms, we propose to split the paths into a number of branches corresponding to most probable codewords of the codes obtained from the Plotkin decomposition. For the sake of concreteness, we consider application of the tree-trellis Viterbi algorithm for finding these codeword. However, any other algorithm, which can successively obtain codewords in the decreasing order of their probability, can be used.

The idea of recursive decomposition of a polar code and ML decoding of outer codes was suggested in [5]–[7] in the context of SC decoding. In this paper we extend this approach to the case of the sequential decoding algorithm, which was shown to have very low decoding complexity and performance comparable to that of list decoding.

## II. Background

### A. Polar codes and the sequential decoding algorithm

An $(n = 2^m, k, d)$ polar code is a binary linear block code generated by $k$ rows of matrix $G_n = B_n A^{\otimes m}$, where $A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $\otimes m$ denotes $m$-fold Kronecker product of a matrix with itself, $B_n$ is an $n \times n$ bit reversal permutation matrix, and $d$ is the minimum distance of the code. Let $c_0^{n-1} = (c_0, \ldots, c_{n-1})$. Any codeword of the polar code can be obtained as $c_0^{n-1} = u_0^{n-1} G_n$, where $u_0^{n-1}$ is the input sequence, such that $u_i = 0, i \in \mathcal{F}$, where $\mathcal{F} \subset \{0, \ldots, n-1\}$ is the set of $n - k$ indices of frozen bit subchannels. The remaining elements of $u_0^{n-1}$ are set to the payload data. It was suggested in [8] that $u_i, i \in \mathcal{F}$ can be set to some linear function of $u_0^{i-1}$(dynamic frozen symbols). This enables one to obtain a code with higher minimum distance than classical polar codes, which can be still efficiently decoded by the SC decoding algorithm and its variations.

978-1-4673-6540-6/15/$31.00 

The sequential decoding algorithm explores paths within the code tree. A path of length $i$ is identified by values $u_0^{i-1}$. Each path is associated with a score. All paths are stored in a stack (priority queue) of size $\Theta$. At each iteration the decoder extends path $u_0^{i-1}$ with the highest score, and performs the $i$-th phase of SC decoding. If the number of elements in the stack exceeds the limit $\Theta$, low score paths are killed. Furthermore, if the decoder returns to phase $i$ more than $L$ times, all paths shorter than $i + 1$ are also killed.

Path score $T(u_0^i, y_0^{n-1})$ is defined as an estimate of the most likely codeword probability. At phase $i$ it can be computed as

$$T(u_0^i, y_0^{n-1}) = \ln R(u_0^i, y_0^{n-1}) + \hat{\Omega}_{\ln}(i) \tag{1}$$

where $R(u_0^i, y_0^{n-1}) = \max_{u_{i+1}^{n-1}} P(u_0^{n-1}|y_0^{n-1})$ and $\hat{\Omega}_{\ln}(i) = \sum_{j \in \mathcal{F}, j>i} \ln(1-P_j)$, $P_j$ is the $j$-th subchannel error probability, provided that exact values of all previous bits $u_{j'}, j' < j$ are available.

### B. Trellis representation

For any linear $(N, K)$ block code $\mathcal{C}$ over $GF(q)$ there exists a minimal span generator matrix $\mathcal{G}$ which can be used to construct a minimal trellis $\mathcal{T}$ for the code [9]. A trellis is an undirected weighted graph, which consists of $N + 1$ non-intersecting subsets of vertices (states), called *levels*, and $N$ non-intersecting subsets of edges. Trellis representation together with the Viterbi algorithm can be used to implement ML decoding of any linear code. The complexity of this method is proportional to the number of edges in the trellis.

## III. Block sequential decoding

### A. Code decomposition

Let us consider an $(n = 2^l, k)$ polar code $\mathcal{C}$ with set $\mathcal{F}_0$ of frozen symbols indices as a code obtained via Plotkin concatenation of polar codes $\mathcal{C}_0$ and $\mathcal{C}_1$, i.e.

$$\mathcal{C} = \{(u + v|u) | u \in \mathcal{C}_0, v \in \mathcal{C}_1\},$$

where $\mathcal{C}_i$ is given by the set of frozen symbol indices $\mathcal{F}_i$, such that $\mathcal{F}_0 = \mathcal{F} \cap \{0, \ldots, n/2-1\}, \mathcal{F}_1 = \{j | j + n/2 \in \mathcal{F}\}$. Similarly to [5], we construct a tree of codes. The leaves of this tree correspond to codes which allow efficient ML decoding. In our implementation, these are codes of dimension at most

2, or rate 1, or length at most $2^t$, where $t$ is a parameter of the proposed algorithm. Every non-leaf node of this tree corresponds to a code $\mathcal{C}_i$, and two its children correspond to codes $\mathcal{C}_{i0}$ and $\mathcal{C}_{i1}$ obtained from its Plotkin decomposition. Each node in this tree is identified by some index $i \in E$, where $E = \cup_{j=0}^{m} \{0,1\}^j$. Codes corresponding to leaves of this tree will be referred to as outer codes. Let $\mathcal{L} \subset E$ be the set of indices $i$ of leaves. Let $I$ be the array of leaf indices $i$ arranged lexicographically in the ascending order. Let $\mathcal{V}$ be the number of leaves in the tree.

Essentially, the SC algorithm recursively decomposes $(n,k)$ polar code $\mathcal{C}$, until it obtains codes of length 1. Each of these codes corresponds to some $u_\phi, 0 \le \phi < n$, where $\phi$ will be referred to as a phase number. Hence, SC decoding of $(n_i,k_i)$ code $\mathcal{C}_i, i \in E$, starts at phase $\phi_i - n_i + 1$ and ends at phase $\phi_i$, so that $\phi_{i1} = \phi_i$, $\phi_{i0} = \phi_i - n_i/2$, and $\phi = n-1$. In the proposed algorithm we employ ML instead of SC decoding for codes $\mathcal{C}_i, i \in \mathcal{L}$.

**Statement 1.** *For any $i$ corresponding to a node in the code decomposition tree $\phi_i \equiv \sum_{j=0}^{m_i-1} 2^j \bmod n_i$, where $n_i = 2^{m_i}$.*

*Proof:* The statement is obvious for $i = \epsilon$, i.e. root node of the tree. Assume that it holds for some node $i$, i.e. $\phi_i = qn_i + \sum_{j=0}^{m_i-1} 2^j$. Since $n_{i1} = n_{i0} = n_i/2$, one obtains $\phi_{i1} \equiv \sum_{j=0}^{m_{i1}-1} 2^j \bmod n_{i1}$, and $\phi_{i0} = \phi_i - 2^{m_i-1} = 2qn_{i0} + \sum_{j=0}^{m_i-2} 2^j$. ■

In the case of polar codes with dynamic frozen symbols we employ the generalized Plotkin decomposition [10], i.e. construct for a $(n_i,k_i)$ code the generator matrix

$$\tilde{G} = \begin{pmatrix} I_{k_{i0}} & 0 & \tilde{I} \\ 0 & I_{k_{i1}} & 0 \end{pmatrix} \begin{pmatrix} G_{i0} & 0 \\ G_{i1} & G_{i1} \\ G_{i2} & G_{i2} \end{pmatrix}, \tag{2}$$

where $I_l$ is a $l \times l$ identity matrix, $G_{ij}, 0 \le j \le 2$, are $k_j \times \frac{n_i}{2}$ matrices, $k_i = k_{i0} + k_{i1}$, and $\tilde{I}$ is obtained by stacking $I_{k_{i2}}$ and a $(k_{i0} - k_{i2}) \times k_{i2}$ zero matrix, where $k_{i2} \le k_{i0}$. Here matrices $G_{i0}, G_{i1}$ generate codes $\mathcal{C}_{i0}, \mathcal{C}_{i1}$, so that decoding of code $\mathcal{C}_i$ requires one first to decode code $\mathcal{C}_{i0}$ to obtain some codeword $xG_{i0}$, and then proceed with decoding in a coset $\mathcal{C}_{i1} + xG_{i2}$.

### *B. The algorithm*

The proposed decoding algorithm employs data structures and path management techniques similar to those introduced in [3], which were modified in [4] for the case of sequential decoding. Similarly to these works, we define real array $S_{l,\lambda}[\beta], 0 \le \lambda \le m, 0 \le \beta < 2^{m-\lambda}, 0 \le l < \Theta$, and boolean array $C_{l,\lambda}[\beta][s], 0 \le s < 2$. In the actual implementation, it is sufficient to take $\lambda \le m - t$. Here $\lambda$ and $\beta$ denote layer and branch number, respectively. Furthermore, we define array of state variables $\mathcal{Z}_{l,i}, i \in \mathcal{L}$. These variables are used to successively obtain codewords of $\mathcal{C}_i$ corresponding to some input LLR vector in the descending order of their probability. We also employ function $RecursivelyCalcS(l,\lambda,\lfloor\phi/2^{m-\lambda}\rfloor)$, which computes $S_{l,\lambda}$ from $S_{l,\lambda-1}$, and $RecursivelyUpdateC(l,\lambda,\lfloor\phi/2^{m-\lambda}\rfloor)$, which computes $C_{l,\lambda-1}$ from $C_{l,\lambda}$ at phase $\phi$ [3], [4].

At phase $\phi$ the sequential algorithm [4] extends path $u_0^{\phi-1}$ into one or two paths $u_0^{\phi}$, so that $C_{l,m}[0][\phi \bmod 2] = u_\phi$. We propose to process jointly symbols, which correspond to a single outer code defined in Section III-A. That is, the proposed decoding algorithm passes only through phases $\phi_i, i \in \mathcal{L}$. During all passes through phase $\phi_i$ the decoder extracts $\chi_i \le \xi_i = \min(2^{k_i}, \Lambda)$ most probable codewords of outer code $\mathcal{C}_i$ corresponding to LLR values $S_{l,m-m_i}[j], 0 \le j < n_i = 2^{m_i}$. The path is cloned $\chi_i - 1$ times, and symbols $c_j$ of the obtained codewords are stored in the corresponding entries $C_{l,m-m_i}[j][z \bmod 2]$, $0 \le j < n_i$, where $z = \lfloor\phi_i/2^{m_i}\rfloor$, and the metrics of the corresponding paths in the code tree are computed and stored in the stack. Furthermore, some state information is preserved in variable $\mathcal{Z}_{l,i}$, so that the next most probable codewords of $\mathcal{C}_i$ corresponding to LLRs $S_{l,m-m_i}[j]$ can be later identified.

The metrics of the paths corresponding to the obtained codewords should be computed so that they are equal to what would be obtained by the sequential decoding algorithm if it takes the corresponding path in the code tree.

At phase $\phi$ the score $T$ of path $l$ is defined as [4]

$$T = (1 - C_{l,m}[0][\phi \bmod 2]) S_{l,m}[0] + D_{l,m} + \hat{\Omega}_{\ln}(\phi), \tag{3}$$

where $S$ is given by

$$S_{l,\lambda}[\beta] = \begin{cases} Q(S_{l,\lambda-1}[2\beta], S_{l,\lambda-1}[2\beta+1]), \ \phi \text{ even}, \\ (-1)^{C_{l,\lambda}[\beta][0]} S_{l,\lambda-1}[2\beta] + S_{l,\lambda-1}[2\beta+1], \ \phi \text{ odd}, \end{cases} \tag{4}$$

where $Q(a,b) = \text{sign}(a)\text{sign}(b)\min(|a|,|b|)$,

$$D_{l,\lambda} = \begin{cases} D_{l,\lambda-1} + \sum_{j=0}^{2^{m-\lambda}-1} Z_{l,\lambda}[j], & 0 < \lambda \le m \\ 0, & \lambda = 0 \end{cases} \tag{5}$$

and $Z_{l,\lambda}[\beta]$ is defined as

$$Z_{l,\lambda}[\beta] = \begin{cases} \max(S_{l,\lambda-1}[2\beta], S_{l,\lambda-1}[2\beta+1]), & \phi \text{ even}, \\ C_{l,\lambda}[\beta][0] S_{l,\lambda-1}[2\beta], & \phi \text{ odd}. \end{cases} \tag{6}$$

For odd $\phi$ one should update the array $C$ according to

$$\begin{aligned} C_{l,\lambda-1}[2\beta][\psi \bmod 2] &= C_{l,\lambda}[\beta][0] \oplus C_{l,\lambda}[\beta][1], \\ C_{l,\lambda-1}[2\beta+1][\psi \bmod 2] &= C_{l,\lambda}[\beta][1], \end{aligned} \tag{7}$$

where $\psi = \lfloor\phi/2\rfloor$.

In order to compute the metrics of the paths obtained at phase $\phi_i$, we need to expand (3) $m_i$ times taking into account (4)–(6). Statement 1 implies that we need to consider only the case of odd $\phi$. One-level expansion of (3) results in

$$\begin{aligned} T =& ((1 - C_{l,m}[0][1])(-1)^{C_{l,m}[0][0]} + C_{l,m}[0][0]) S_{l,m-1}[0] + \\ & (1 - C_{l,m}[0][1]) S_{l,m-1}[1] + D_{l,m-1} + \hat{\Omega}_{\ln}(\phi_i) \end{aligned} \tag{8}$$

It is easy to see that $(1 - C_{l,m}[0][1])(-1)^{C_{l,m}[0][0]} + C_{l,m}[0][0]$ is equal to $1 - (C_{l,m}[0][0] \oplus C_{l,m}[0][1])$. Using (7), one can write (8) as

$$\begin{aligned} T =& (1 - C_{l,m-1}[0][\lfloor\phi_i/2\rfloor \bmod 2]) S_{l,m-1}[0] \\ & + (1 - C_{l,m-1}[1][\lfloor\phi_i/2\rfloor \bmod 2]) S_{l,m-1}[1] \\ & + D_{l,m-1} + \hat{\Omega}_{\ln}(\phi_i). \end{aligned} \tag{9}$$

Performing this expansion $m_i$ times, one obtains

$$T = M_{l,m-m_i} + D_{l,m-m_i} + \hat{\Omega}_{\ln}(\phi_i). \tag{10}$$

where

$$M_{l,m-m_i} = \sum_{j=0}^{n_i-1} (1 - C_{l,m-m_i}[j][z \bmod 2]) S_{l,m-m_i}[j],$$

and $z = \lfloor \phi_i / n_i \rfloor$. The values $S_{l,m-m_i}[j]$ can be computed by calling $RecursivelyCalcS(l, m - m_i, z)$, where $l$ is the number of the path (see [4] for implementation details). It will be assumed that the decoders of outer codes compute scores $F = M_{l,m-m_i} + D_{l,m-m_i}$ for each obtained codeword. This requires $D_{l,m-m_i}$ to be included into the state variable $\mathcal{Z}_{l,i}$.

Hence, the proposed block sequential decoding algorithm (BSDA) operates as follows:

1) Let $l_0$ be the index of the initial path. Let $S_{l_0,0}[j]$ be the received symbol LLRs. Let $J[l_0] = 0$. Put $(0, l_0)$ into the stack. Let $\chi_{l_0,i} = 0, q_i = 0, i \in \mathcal{L}$, $P = 1$.
2) Extract $(M, l)$ with the highest $M$ from the stack. If $J[l] \geq \mathcal{V}$, return $C_{l,0}[\beta][0], 0 \leq \beta < n$, and terminate. Let $P = P - 1$.
3) Let $i' = I[J[l] - 1]$. If $\chi_{l,i'} < \xi_{i'}$, extract the next most probable codeword $c$ of $\mathcal{C}_{i'}$, together with the corresponding value $F$, using the state variable $\mathcal{Z}_{l,i'}$. Clone the current path, and let $C_{l',m-m_{i'}}[\beta][z' \bmod 2] = c_\beta$, where $z' = \lfloor \phi_{i'} / n_{i'} \rfloor$ and put $(F + \hat{\Omega}_{ln}(\phi_{i'}), l')$ into the stack, where $l'$ is the index of the cloned path. Let $P = P + 1$, $\chi_{l,i'} = \chi_{l,i'} + 1$.
4) Let $i = I[J[l]]$, $z_i = \lfloor \phi_i / n_i \rfloor$. Let $q_i = q_i + 1$. Compute $S_{l,m-m_i}[\beta], 0 \leq \beta < n_i$, and $\delta = D_{l,m-m_i}$ by calling $RecursivelyCalsS(l, m - m_i, z_i)$.
5) Preprocess $\delta$ and LLR values $S_{l,m-m_i}[\beta]$ using a decoder of the outer code $\mathcal{C}_i$. Save the decoder state (i.e. the results of pre-processing) in variable $\mathcal{Z}_{l,i}$. Then use it to obtain $1 \leq \tau \leq \xi_i$ most probable codewords $c^{(s)}$ together with the corresponding scores $F^{(s)}$ , $0 \leq s < \tau$. Let $\chi_{l,i} = \chi_{l,i} + \tau$.
6) If $P + \tau > \Theta$, kill $P + \tau - \Theta$ paths with the smallest metrics, remove them from the stack and decrease $P$ appropriately.
7) For each $s$ do: if $s > 0$, clone path $l$ to obtain new path $l'$, and let $l' = l$ otherwise. Let $C_{l',m-m_i}[\beta][z_i \bmod 2] = c_\beta^{(s)}$. Put $(F^{(s)} + \hat{\Omega}_{ln}(\phi_i), l')$ into the stack. If $z_i \equiv 1 \bmod 2$, call $RecursivelyUpdateC(l', m - m_i, z_i)$. Let $J[l'] = J[l] + 1$.
8) Let $P = P + \tau$, $\chi_i = \chi_i + \tau$.
9) If $q_i > L$, kill all paths $l' : J[l'] < J[l]$, remove them from the stack and decrease $P$ appropriately.
10) Go to step 2.

The decoders of outer codes must be able to compute one or more most probable codewords corresponding to a given LLR vector (see step 5). Furthermore, they must preserve some state information, which can be used later at step 3 to obtain next most probable codewords. Observe that step 3 is invoked only if the block sequential algorithm has accessed the last most probable codeword of $\mathcal{C}_i$ obtained so far.

It is possible to slightly reduce the complexity of the proposed algorithm if one does not retrieve at steps 3 and 5 codewords $c^{(s)}$ with $F^{(s)} < F^{(0)} - \Delta$ for some $\Delta > 0$.

The proposed algorithm preserves the property of algorithms [3], [4], [11], that at most $L$ paths pass through each phase $\phi_i$. Therefore, the maximal number of iterations performed by the decoder is upper bounded by $\mathcal{V}L$. The parameter $\Lambda$ affects the performance, complexity and amount of memory needed to store state variables $\mathcal{Z}_{l,i}$. The parameter $\Theta \leq \mathcal{V}L$ determines the amount of memory used.

### C. Decoding of outer codes

*1) Tree-trellis Viterbi algorithm:* We propose to employ the tree-trellis Viterbi list algorithm (TTA) [12], [13]. It includes forward pass (preprocessing), which computes partial path metrics, and backward pass, which retrieves codewords in the decreasing order of their correlation metric. This algorithm can find $\Lambda$ codewords successively, so that the $s$-th codeword $c^{(s)}$ is obtained using previously discovered codewords $c^{(s')}, s' < s \leq \Lambda$. Observe that the multiple-list tree trellis algorithm [12] has better asymptotic complexity, but for the considered case of decoding very short outer codes it induces too much overhead.

The original TTA was described for the case of convolutional codes. For completeness, we present a brief description of this algorithm with appropriate changes needed to accomodate the case of linear block codes, and extraction of at most $\xi$ codewords with metric $F^{(s)} \geq F^{(0)} - \Delta$.

Let $\sigma_t, 0 \leq t < n$, be the input LLRs, and consider decoding of $(n, k)$ linear code $\mathcal{C}$. Decoding is performed using the minimal trellis of $\mathcal{C}$.

A *path* $p = p(0), p(1), \ldots, p(n)$ of length $n$ is a sequence of connected trellis nodes $p(i), i = 0, \ldots, n$, which describes a codeword of $\mathcal{C}$. Let $p_s$ be the $s$-th best path. In the forward pass of TTA, for each node $(t, j)$, where $t$ and $j$ denote level and state, respectively, two partial path metrics $M_1[t][j]$ and $M_2[t][j]$, $M_1[t][j] \geq M_2[t][j]$ are stored. The metric $M_1[t][j]$ is the smallest partial path metric from node $(0, 0)$ to node $(t, j)$. It can be obtained as in the Viterbi algorithm with

$$M_1[t][j] = \max_i \{M_1[t-1][i] + M(i \to j, t)\}, \ t > 0, \quad (11)$$

where $M(i \to j, t) = \sigma_{t-1}(1 - r(i \to j, t))$, and $r(i \to j, t)$ is the codeword symbol corresponding to transition from $(t-1, i)$ to $(t, j)$. Observe that in the minimal trellis there are at most two nodes $(t-1, i)$ connected by edges with node $(t, j)$. Let $v[t][j]$ be the state that gives the maximum in (11), and $\bar{v}[t][j]$ be the other state. If there is only one node having a transition to $(t, j)$, then let $\bar{v}[t][j] = v[t][j]$. Let us further define

$$M_2[t][j] = \begin{cases} +\infty, & t = 0, \\ M_1[t-1][\bar{v}[t][j]] + M(\bar{v}[t][j] \to j, t), & t > 0. \end{cases}$$

It is the smallest partial path metric from node $(0, 0)$ to node $(j, t)$ via node $(\bar{v}[t][j], t-1)$.

The backward passes use a stack (priority queue) $\mathcal{S}$ of elements $S = (S.m, S.q, S.t, S.u), h \geq 1$, where:

- $S.m$ is a path metric;
- $S.q$ is the serial number of the call to the backward pass algorithm, at which $S$ was inserted into the stack;
- $S_t$ is a level;
- $S.u$ is a partial path metric from node $(p_{S_q}(S.t), S.t)$ to node $(n, 0)$

BACKWARDPASS()

1 **if** $s > \xi$
2 **then return** $NULL$
3 **if** $s = 0$
4 **then** $i \leftarrow 0; t \leftarrow n; m \leftarrow 0; p_s[t] \leftarrow 0;$
5 **else** $S \leftarrow \mathcal{S}$.POPMAX();
6 $t \leftarrow S.t; i \leftarrow p_{S.q}[t]; m \leftarrow S.u;$
7 **for** $(l = t, \ldots, n)$
8 **do** $p_s[l] \leftarrow p_{S.q}[l]$
9 $j \leftarrow \bar{v}[t][i]$
10 $m \leftarrow m + M_2[t][i] - M_1[t-1][j]$
11 $p_s[t-1] \leftarrow j; t \leftarrow t-1; i \leftarrow j$
12 **while** $t \neq 0$
13 **do if** $v[t][i] \neq \bar{v}[t][i]$
14 **then** $\mathcal{S}$.PUSHBACK$(S(M_2[t][i] + m, s, t, m))$
15 $j \leftarrow v[t][i]$
16 $m \leftarrow m + M_1[t][i] - M_1[t-1][j]$
17 $p_s[t-1] = j$
18 $t \leftarrow t-1; i \leftarrow j$
19 **if** $s = 0$
20 **then** $edge \leftarrow m - \Delta$
21 **else if** $m < edge$
22 **then return** $\emptyset$
23 $s \leftarrow s + 1$
24 Let $C$ be the codeword corresponding to path $p_s$
25 **return** $(C, m + \delta)$

Fig. 1: TTA backward pass.

The stack provides a function $PopMax$, which returns the element with the highest $S.m$, and removes it from the stack.

The preprocessing operation used at step 5 of the above described BSDA for polar codes corresponds to the forward pass of the TTA. This operation involves computing metrics $M_1[t][i]$ and $M_2[t][i]$. One should also set $s = 0$ and save the value of $\delta$. In order to extract codewords in the decreasing order of their probability, the algorithm shown in Figure 1 should be used. For $s = 0$ it follows the most probable path in the trellis. During each pass over the trellis, it stores in the stack at line 14 the metrics of paths which diverge from the current one. Therefore, each subsequent call results in a codeword with smaller metric $m$. If the algorithm returns $\emptyset$, the corresponding operations of the BSDA should be skipped.

In order to use this algorithm in the BSDA, one needs to save the contents of the stack $\mathcal{S}$, the value of $\delta$, as well as $M_1[t][j], M_2[t][j], v[t][j], p_s[t]$ in the corresponding state variable $\mathcal{Z}_{l,i}$. Note that $p_s[t]$ can be stored in a compact form, as described in [12].

The complexity of the forward pass of TTA is $O(e)$, where $e$ is the number of edges in the trellis. The complexity of the backward pass is $O(n\xi \log \xi)$.

*2) Simplified decoding:* For some codes it is possible to design a simpler decoding algorithm. In the case of $k_i \leq 2$ it is easy to produce all $2^{k_i}$ codewords. In the case of $k_i = n_i$ one can enumerate most probable vectors using the techniques presented in [14], [15]. In practice it is sufficient to find 4 such vectors. This can be done in $O(n_i)$ operations.

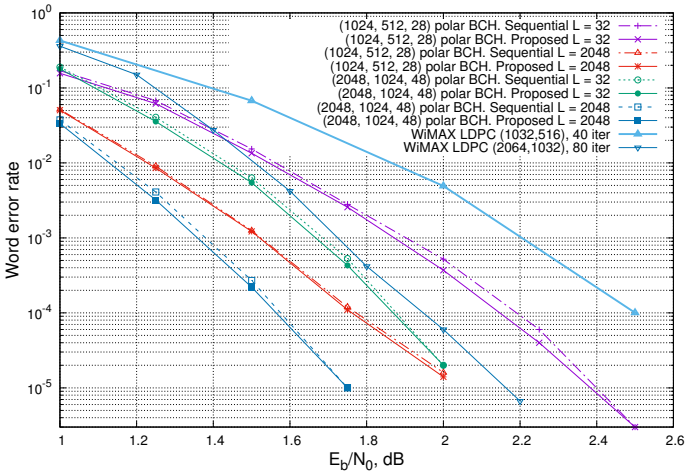


Fig. 2: Performance of the decoding algorithms

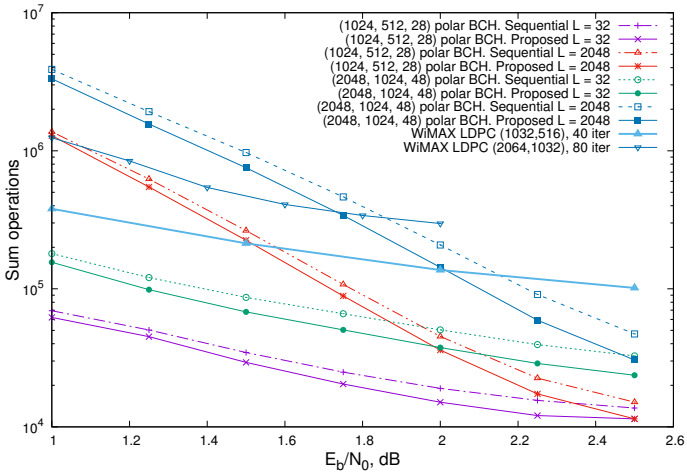


Fig. 3: Average number of summations

## IV. NUMERICAL RESULTS

Fig. 2 present simulation results illustrating the performance of the proposed decoding algorithm compared to the sequential decoding algorithm for the case of BPSK transmission over AWGN channel. The results are reported for $(1024, 512, 28)$ and $(2048, 1024, 48)$ polar codes with dynamic frozen symbols obtained as subcodes of extended BCH codes [8], and various values of list size $L$. For comparison, we report also results for similar WiMAX LDPC codes. We have found that in order to obtain the performance comparable to that of the sequential decoding algorithm for the considered codes, while minimizing the complexity of the proposed algorithm, one can set $t = 3$ (i.e. TTA is used only for outer codes of length 8), $\Lambda = 30$ and $\Delta = 30$. It can be seen that the proposed algorithm provides slightly better performance compared to the sequential algorithm.

Fig. 3 and Fig. 4 present average decoding complexity for both algorithms with the above parameters. It can be seen that the average number of summation and comparison operations in the case of the proposed algorithm is $20 - 35\%$ and $50\%$

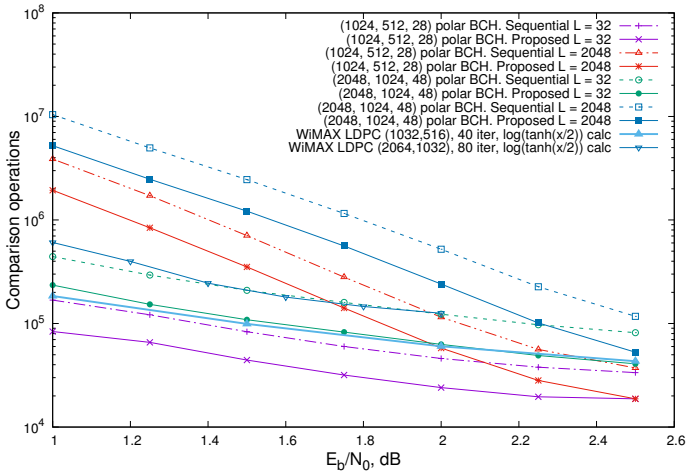


Fig. 4: Average number of comparison operations

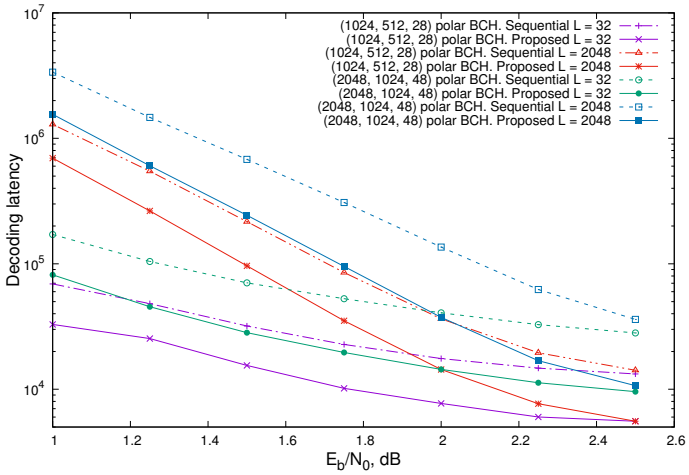


Fig. 5: Average number of clock cycles

less than in the case of the sequential algorithm, respectively. Furthermore, the average number of operation with the stack (priority queue) is reduced by a factor of ten. It can be also seen that at $E_b/N_0 = 1.7$ dB the average number of summations requires by the decoders of LDPC and polar codes of length $n \approx 2048$ is approximately the same, but the polar code provides 100 times lower decoding error probability. The number of comparisons in the case of polar codes exceeds the number of calls to $\log\tanh(x/2)$ needed by the belief propagation algorithm. However, the latter operation is much more expensive, even if implemented approximately.

Figure 5 presents the average number of clock cycles for the sequential and the proposed decoding algorithms. It can bee seen that average decoding latency is reduced by a factor of three. This comes from the reduced depth of the recursion in $RecursivelyUpdateC$ and $RecursivelyCalcS$, which correspond to the implementation of the SC algorithm. The latency of TTA can be seen to be much less than that of the SC algorithm. However, the obtained gain is somewhat less than that achieved in [16]. Some extra clock cycles are spent by the proposed algorithm on operations with the stack.

## V. Conclusion

In this paper a modification of the sequential decoding algorithm was proposed. It enables one to reduce the average decoding complexity and latency without any performance loss. This is achieved by recursive representation of the polar code as a Plotkin concatenation of some outer codes. This enables one to process jointly symbols, which correspond to a single outer code, by employing some ML decoding algorithm. In this paper we considered application of the tree-trellis Viterbi algorithm for decoding of outer codes, which is able to successively retrieve codewords in the decreasing order of their probability. For short outer codes this algorithm has lower complexity and latency compared to the successive cancellation algorithm and its variations. However, its complexity grows exponentially with code length. It may be possible to apply more efficient near-ML decoding algorithms for decoding of outer codes.

## References


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

[2] I. Dumer and K. Shabunov, "Soft-decision decoding of ReedMuller codes: Recursive lists," *IEEE Trans. on Inf. Theory*, vol. 52, no. 3, March 2006.

[3] I. Tal and A. Vardy, "List decoding of polar codes," in *Proceedings of IEEE International Symposium on Information Theory*, 2011, pp. 1–5.

[4] V. Miloslavskaya and P. Trifonov, "Sequential decoding of polar codes," *IEEE Communications Letters*, vol. 18, no. 7, pp. 1127–1130, 2014.

[5] G. Sarkis and W. Gross, "Increasing the throughput of polar decoders," *IEEE Communications Letters*, vol. 17, no. 4, pp. 725–728, April 2013.

[6] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal On Selected Areas In Communications*, vol. 32, no. 5, May 2014.

[7] B. Li, H. Shen, D. Tse, and W. Tong, "Low-latency polar codes via hybrid decoding," in *Proceedings of International Symposium on Turbo Codes and Iterative Information Processing*, 2014.

[8] P. Trifonov and V. Miloslavskaya, "Polar codes with dynamic frozen symbols and their decoding by directed search," in *Proceedings of IEEE Information Theory Workshop*, September 2013, pp. 1 – 5.

[9] A. Kiely, J. Dolinar, S.J., R. McEliece, L. Ekroot, and W. Lin, "Trellis decoding complexity of linear block codes," *IEEE Transactions On Information Theory*, vol. 42, no. 6, pp. 1687–1697, November 1996.

[10] P. Trifonov and V. Miloslavskaya, "Twisted polar codes," in *Proc. of Int. Symp. on Information Theory and Its Applications*, 2014, pp. 456–460.

[11] K. Chen, K. Niu, and J. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Trans. on Communications*, vol. 61, no. 8, pp. 3100–3107, August 2013.

[12] M. Roder and R. Hamzaoui, "Fast tree-trellis list Viterbi decoding," *IEEE Trans. on Communications*, vol. 54, no. 3, pp. 453–461, March 2006.

[13] F. K. Soong and E.-F. Huang, "A tree-trellis based fast search for finding the $n$ best sentence hypotheses in continuous speech recognition," in *Proceedings of IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991.

[14] A. Valembois and M. Fossorier, "An improved method to compute lists of binary vectors that optimize a given weight function with application to soft-decision decoding," *IEEE Communications Letters*, vol. 5, no. 11, pp. 456–458, Nov 2001.

[15] T. Kusaka, "Introduction of a recursive method for specific weight binary vector generation in decreasing order of its reliability measure," in *Proc. of Int. Symp. on Information Theory and Its Applications*, 2014.

[16] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," in *IEEE Workshop on Signal Processing Systems, 2014*, October 2014, pp. 1–6.