Fast Block Sequential Decoding of Polar Codes

Grigorii Trofimiuk, *Student Member, IEEE,* Nikolai Iakuba, *Student Member, IEEE,* Stanislav Rets, Kirill Ivanov, *Student Member, IEEE,* Peter Trifonov, *Member, IEEE*

***Abstract*—A reduced complexity sequential decoding algorithm for polar (sub)codes is described. The proposed approach relies on a decomposition of the polar (sub)code being decoded into a number of outer codes, and on-demand construction of codewords of these codes in the descending order of their probability. Construction of such codewords is implemented by fast decoding algorithms, which are available for many codes arising in the decomposition of polar codes. Further complexity reduction is achieved by taking hard decisions of the intermediate LLRs, and avoiding decoding of some outer codes. Data structures for sequential decoding of polar codes are described.**

The proposed algorithm can be also used for decoding of polar codes with CRC and short extended BCH codes. It has lower average decoding complexity compared with the existing decoding algorithms for the corresponding codes.

***Index Terms*—Polar codes, polar subcodes, sequential decoding, Plotkin construction.**

I. Introduction

Polar codes are first capacity-achieving codes with low-complexity construction, encoding and decoding methods [1]. Near maximum likelihood (ML) decoding can be performed with the Tal-Vardy successive cancellation list (SCL) decoding [2]. However, finite-length performance of polar codes is quite poor, motivating thus development of improved constructions. Short polar subcodes [3], [4] and CRC-aided polar codes [2] were shown to outperform state-of-the-art LDPC and turbo codes under list decoding with small list size.

The complexity of SCL decoding algorithm can be reduced by employing block decoding, i.e. joint processing of subsequent blocks of information symbols [5], [6], [7], [8], [9], [10], or symbol-based decoding techniques [11]. The complexity of this method can be further reduced by constructing unrolled decoders with simplified flow control logic [12].

Another approach is to utilize stack decoding [13] or its improved version known as the sequential decoding algorithm (SDA) [14], [15]. This algorithm avoids construction of many useless low-probability paths in the code tree. For sufficiently high SNR, its complexity approaches that of the successive cancellation (SC) decoding algorithm with the performance close to that of the SCL method. Varied improvements of stack decoding were also proposed in [16], [17], [18], [19].

This work was supported by Government of Russian Federation (grant 08-08).

This work was partially presented at the International Workshop on Wireless Communication Systems'2015.

G. Trofimiuk, N. Iakuba and P. Trifonov are with ITMO University, Saint-Petersburg, Russia. K. Ivanov is with EPFL, Switzerland. E-mail: {gtrofimiuk,nyakuba,pvtrifonov}@itmo.ru, kirill.ivanov@epfl.ch

An alternative way to implement decoding of polar codes is based on sphere decoding [20], [21], [22], [23]. However, the complexity of this method grows quickly with code length, so that the results for this method have been reported only for very short codes.

The idea of joint processing of some blocks of information symbols was suggested in [5] in the context SC decoding and generalized in [7] for the case of SCL decoding. In this paper we extend this approach to the case of sequential decoding. We show that the proposed method, referred to as block sequential decoding algorithm (BSDA), can provide the performance close to that of the SC list decoder with large list size with complexity approaching (at high SNR) that of the unrolled SC decoder. The proposed approach can be used both for polar subcodes and polar codes with CRC. Furthermore, we show that, by exploiting the representation of a linear code via a system of dynamic freezing constraints, the proposed approach can be used for decoding of other error-correcting codes. In particular, we show that for a $(128, 64, 22)$ extended primitive narrow-sense BCH (eBCH) code the proposed algorithm provides better performance and lower complexity compared with a recent trellis-based sequential-type algorithm [24].

The paper is organized as follows. The background on polar codes and the decoding algorithms is presented in Section II. The BSDA is introduced in Section III. The algorithms for decoding of some outer codes are presented in Section IV. The construction of polar subcodes and its processing in the proposed algorithm is discussed in Section V. Complexity analysis is provided in Section VI. Simulation results are presented in Section VII. The implementation details of low-level data structures are described in the Appendix.

II. Background

A. Polar codes and Plotkin decomposition

For a positive integer n, denote by $[n]$ the set of n integers $\{0, 1, \dots n-1\}$. An $(n = 2^m, k)$ polar code over $\mathbb{F}_2$ is a linear block code generated by k rows of matrix[1] $A_m = F^{\otimes m}$, where $F = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $\otimes m$ denotes m-times Kronecker product of the matrix with itself [1]. Hence, a codeword of a classical polar code is obtained as $c_0^{n-1} = u_0^{n-1} A_m$, where $u_s^t = (u_s, u_{s+1}, \dots, u_t)$, $u_i = 0, i \in \mathcal{F}$, $\mathcal{F} \subset [n]$ is the set of $n-k$ frozen symbol indices, which will be referred to as *frozen set*, and the remaining symbols are set to the data symbols being encoded.

[1]Polar codes are typically defined with the bit-reversal permutation matrix. However, it is convenient here to omit it, since this results in a simpler description of the proposed decoding algorithm.

An $(n=2^m,k)$ polar code $\mathcal{C}$ with the frozen set $\mathcal{F}$ can be represented as a code obtained via Plotkin concatenation of polar codes $\mathcal{C}_0$ and $\mathcal{C}_1$, i.e.

$$\mathcal{C}=\left\{(u+v|v) \mid u\in\mathcal{C}_0, v\in\mathcal{C}_1\right\}, \tag{1}$$

where $\mathcal{C}_i$ is given by the frozen set $\mathcal{F}_i$, $\mathcal{F}_0=\mathcal{F}\cap[n/2]$, $\mathcal{F}_1=\{j\mid j+n/2\in\mathcal{F}\}$. Such representation will be referred to as *Plotkin decomposition* (PD) of the code $\mathcal{C}$ into the codes $\mathcal{C}_0$ and $\mathcal{C}_1$. This decomposition can be applied recursively. After a number of decomposition steps, it results in some simple codes such as repetition, SPC, etc, which admit low complexity decoding [5], [7].

B. Sequential decoding of polar codes

Consider the decoding of (n,k) polar code. Let u_0^{n-1} be the vector of input symbols of the polarizing transformation used by the transmitter. Given a received noisy vector y_0^{n-1}, the SDA constructs a number of partial candidate vectors (paths) $v_0^{\phi-1}\in\mathbb{F}_2^{\phi},\phi\leq n$, where $\phi-1$ is referred to as a path *phase*, then evaluates how close their continuations v_0^{n-1} may be to the received sequence, and eventually produces a single codeword, being a solution of the decoding problem.

The algorithm makes use of a double-ended priority queue (PQ). A PQ is a data structure, which stores tuples $(M,v_0^{\phi-1})$, where $M=M(v_0^{\phi-1},y_0^{n-1})$ is the score of the path $v_0^{\phi-1}$, and provides efficient algorithms for the following operations [25]:

- push a tuple into the PQ;
- pop a tuple $(M,v_0^{\phi-1})$ (or just $v_0^{\phi-1}$) with the highest M;
- pop a tuple $(M,v_0^{\phi-1})$ (or just $v_0^{\phi-1}$) with the lowest M;
- remove a tuple from the PQ.

We assume here that the PQ may contain at most D elements.

We employ the multilevel bucket PQ implementation [26], which is much more efficient compared to the heap-based approach [16] in the context of sequential decoding.

If the decoder returns to a phase i more than L times, all paths shorter than $i+1$ are also removed. The parameter L affects the performance of SDA in the same way as list size in the SCL decoder.

The stack decoding algorithm for polar codes operates as follows [27]:

1) Push into the PQ a zero-length vector with score 0. Let $q_0^{n-1}=0$, where q_ϕ is the counter for the number of visits to phase ϕ.
2) Extract from the PQ a path $v_0^{\phi-1}$ with the highest score. Let $q_\phi\leftarrow q_\phi+1$.
3) If $\phi=n$, return codeword $v_0^{n-1}A_m$ and terminate the algorithm.
4) If the number of valid continuations v_0^{ϕ} of a path $v_0^{\phi-1}$ exceeds the amount of free space in the PQ, remove from it the element with the smallest score.
5) Compute scores $M(v_0^{\phi},y_0^{n-1})$ of valid continuations v_0^{ϕ} of the extracted path, and push them into the PQ. Let $\phi\leftarrow\phi+1$.
6) If $q_\phi\geq L$, remove from the PQ all paths $v_0^{j-1},j\leq\phi$.
7) Go to step 2.

In what follows, by iteration we mean one pass of the above algorithm over steps 2–7.

The parameter $D\leq kL$ affects the amount of memory needed by the sequential decoder. In general, D can be much less than kL, however, setting D too small may results in performance degradation.

A score function $M(v_0^{\phi},y_0^{n-1})$ can be obtained as a generalization of the Fano metric, which was introduced for sequential decoding of convolutional codes [28]. In the context of polar codes, its approximate version can be written as [15]

$$M(v_0^{\phi-1},y_0^{n-1})=\underbrace{\sum_{i=0}^{\phi-1}\tau(S_m^{(i)}(v_0^{i-1}|y_0^{n-1}),v_i)}_{R(v_0^{\phi-1}|y_0^{n-1})}-\Psi(\phi), \tag{2}$$

where $\Psi(\phi)=\mathbf{E}_{Y_0^{n-1}}\left[R(u_0^{\phi-1}|Y_0^{n-1})\right]$ is the bias function, which can be pre-computed offline, Y_0^{n-1} are the random variables corresponding to the received vector,

$$\tau(S,v)=\begin{cases}0, & \operatorname{sgn}(S)=(-1)^v\\ -|S|, & \text{otherwise},\end{cases}$$

is the penalty function, and $S_m^{(i)}(v_0^{i-1},y_0^{n-1})$ are the modified log-likelihood ratios (LLRs) [15], [29], which are given by

$$S_\lambda^{(2i)}(v_0^{2i-1},y_0^{2^\lambda-1})=Q(a,b)=\operatorname{sgn}(a)\operatorname{sgn}(b)\min(|a|,|b|), \tag{3}$$

$$S_\lambda^{(2i+1)}(v_0^{2i},y_0^{2^\lambda-1})=P(v_{2i},a,b)=(-1)^{v_{2i}}a+b, \tag{4}$$

where λ is a layer, $a=S_{\lambda-1}^{(i)}(v_{0,e}^{2i-1}\oplus v_{0,o}^{2i-1},y_{0,e}^{2^\lambda-1})$, and $b=S_{\lambda-1}^{(i)}(v_{0,o}^{2i-1},y_{0,o}^{2^\lambda-1})$.

The first term of (2) is the total penalty of path $v_0^{\phi-1}$ for its deviation from the one given by the hard decisions based on LLRs $S_m^{(i)}(v_0^{i-1}|y_0^{n-1})$. The second term is the expected value of the first term under the assumption that path $v_0^{\phi-1}$ is correct. Bias term allows one to properly compare the paths of different length and results in a huge reduction of the average number of iterations performed by the stack algorithm [15] compared with the original implementation [27].

Similarly to the case of sequential decoding of convolutional codes, the described algorithm does not necessarily perform ML decoding even for $L=\infty$. This is due to the bias term in the path score, which may cause the correct path to be removed, if its score drops too sharply at some early phase ϕ.

III. Block sequential decoding

We propose to reduce the complexity of sequential decoding by joint processing of blocks of input symbols of the polarizing transformation. Similar approach was suggested in [7] in the context of list decoding. However, we show that in the case of sequential decoding this provides some additional benefits. Most importantly, one does not need to construct immediately L most probable codewords for each block. Instead, these codewords can be constructed on-demand, and in many cases just one codeword is sufficient.

A. Recursive decomposition of polar codes

Let us consider decoding of an (n, k) polar code $\mathcal{C}$. We propose to recursively apply PD to the code until one obtains codes which admit efficient decoding. This results in a code decomposition tree similar to that introduced in [5], [30].

Each non-leaf node of this tree corresponds to a code $\hat{\mathcal{C}}$, and two its children correspond to codes $\hat{\mathcal{C}}_0$ and $\hat{\mathcal{C}}_1$ obtained from its PD. Codes corresponding to the leaves of this tree will be referred to as *outer codes*. Let $\mathcal{V}$ be the number of leaves in the tree. We enumerate outer codes $\mathcal{C}_\psi$ with indices $\psi \in [\mathcal{V}]$ in the ascending order from the leftmost to the rightmost leaf of the code decomposition tree (see Figure 1).

Essentially, list and sequential algorithms recursively decompose (n, k) polar code $\mathcal{C}$, until codes of length 1 are obtained. Each of these codes corresponds to symbols $u_\phi, 0 \leq \phi < n$, where ϕ is the phase number. We propose to stop this recursion at some layers, and arrange symbols u_ϕ into blocks, which correspond to $(n_\psi = 2^{m_\psi}, k_\psi, d_\psi)$ codes $\mathcal{C}_\psi, \psi \in [\mathcal{V}]$, obtained via PD, where n_ψ is length, k_ψ is dimension, and d_ψ is minimum distance of $\mathcal{C}_\psi$. The i-th block starts at phase $\phi_\psi - n_\psi + 1$ and ends at phase ϕ_ψ, Symbols within the block are processed jointly. This processing reduces to list decoding of outer codes $\mathcal{C}_\psi$. Construction of such a decomposition can be simplified by employing the techniques suggested in [31].

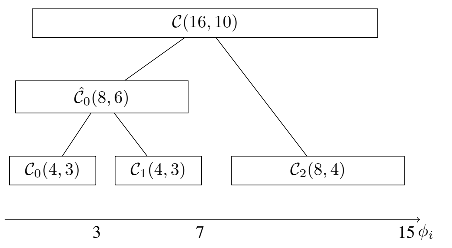


Fig. 1: Plotkin decomposition tree for $(16, 10)$ code

Example 1. *Consider the* $(16, 10)$ *polar code* $\mathcal{C}$ *with frozen set* $\mathcal{F} = \{0, 4, 8, 9, 10, 12\}$.

The PD tree of this code is shown in figure 1. One step of PD results in codes $\hat{\mathcal{C}}_0$ *and* $\mathcal{C}_2$. $(8, 6)$ *code* $\hat{\mathcal{C}}_0$ *is non-leaf in PD tree with the frozen set* $\hat{\mathcal{F}}_0 = \{0, 4\}$. *By applying the PD to the code* $\hat{\mathcal{C}}_0$, *one obtains* $(4, 3)$ *outer codes* $\mathcal{C}_0 = \mathcal{C}_1$ *with frozen sets* $\mathcal{F}_0 = \mathcal{F}_1 = \{0\}$. *These codes can be efficiently decoded (see sections IV-B and IV-C for details). The* $(8, 4)$ *code* $\mathcal{C}_2$ *with* $\mathcal{F}_2 = \{0, 1, 2, 4\}$ *can also be efficiently decoded (see section IV-B), hence we stop the recursion.*

It remains to transform the path score function (2) into a form suitable for use with decoders of outer codes. Let $E(c_0^{n-1}, S_0^{n-1}) = \sum_{i=0}^{n-1} \tau(S_i, c_i)$ be the ellipsoidal weight[2] (also known as correlation discrepancy) of vector $c_0^{n-1} \in \mathbb{F}_2^n$ with respect to LLRs S_0^{n-1} [32], [33].

Lemma 1. *For any* $c_0^{2n-1} \in \mathbb{F}_2^{2n}$ *one has* $E(c_0^{2n-1}, S_0^{2n-1}) = E(c_0^{n-1} \oplus c_n^{2n-1}, \tilde{S}_0^{n-1}) + E(c_n^{2n-1}, \overline{S}_0^{n-1})$, *where* $\tilde{S}_i = Q(S_i, S_{i+n})$, $\overline{S}_i = P(c_i \oplus c_{i+n}, S_i, S_{i+n})$.

[2]Conventionally, it is defined as a non-negative function. However, here we define it as a non-positive one to ensure consistency with the values which arise in the Tal-Vardy algorithm.

Proof. It can be seen that $E(c_0^{2n-1}, S_0^{2n-1}) = \sum_{i=0}^{n-1} E((c_i, c_{i+n}), (S_i, S_{i+n}))$. Hence, it is sufficient to prove the statement for $n = 1$. Observe that $E(c_0^1, S_0^1) = \gamma = E((0, 0), (S'_0, S'_1))$, where $S'_i = (-1)^{c_i} S_i$. Hence, it is sufficient to consider the case of $c_i = 0$.

For the case $Q(S'_0, S'_1) > 0$ one has $\gamma = \tau(S'_0, 0) + \tau(S'_1, 0) = \tau(S'_0 + S'_1, 0)$, while for $Q(S'_0, S'_1) < 0$ one has $\gamma = \min(|S'_0|, |S'_1|) + \tau(S'_0 + S'_1, 0)$. The latter equality follows by considering the cases of $S'_0 + S'_1 > 0$ and $S'_0 + S'_1 \leq 0$. $\square$

Theorem 1. *The ellipsoidal weight of the vector* $u_0^{2^m-1} A_m$ *with respect to the input LLRs* $\mathbf{S}$ *is equal to score of the path* $u_0^{2^m-1}$ *in the SC decoder, i.e.* $E(u_0^{2^m-1} A_m, \mathbf{S}) = \sum_{i=0}^{2^m-1} \tau(S_m^{(i)}(u_0^{i-1}, y_0^{2^m-1}), u_i)$, *where* $\mathbf{S} = (S_0^{(0)}(y_0), \ldots, S_0^{(0)}(y_{2^m-1}))$.

Proof. For $m = 0$, the statement is obvious. Let us assume that it is valid for some $m \geq 0$. Then, from Lemma 1, one obtains $E(u_0^{2^{m+1}-1} A_{m+1}, \mathbf{S}) = E(u_0^{2^m-1} A_m, \widetilde{\mathbf{S}}) + E(u_{2^m}^{2^{m+1}-1} A_m, \overline{\mathbf{S}})$, where $\widetilde{\mathbf{S}}_i = S_1^{(0)}(y_i, y_{i+2^m}), 0 \leq i < 2^m$, and $\overline{\mathbf{S}}_i = S_1^{(1)}((u_0^{2^m-1} A_m)_i, (y_i, y_{i+2^m}))$. Then the result follows from the inductive assumption. $\square$

Theorem 1 implies that the sum of those terms in (2), which correspond to the same block in the PD tree, can be obtained by construction of a codeword of the corresponding outer code $\mathcal{C}_\psi$, and computing its ellipsoidal weight. In some cases this can be more efficient than performing iterations of SDA.

Observe that the proposed approach can be also viewed as follows: we stop the calculation of LLRs $S_m^{(\phi_\psi)} = S_m^{(\phi_\psi)}(v_0^{\phi_\psi - 1}, y_0^{n-1})$, given by recursion (3)–(4), at layer $m - m_\psi$ and decode 2^{m_ψ} LLRs $\mathbf{S} = S_{m-m_\psi}^{(r(\psi))}$ in code $\mathcal{C}_\psi$, where $r(\psi) = \lfloor \phi_\psi / 2^{m_\psi} \rfloor$. Thus, Theorem 1 allows one to rewrite the score (2) for the path constructed up to block ψ as

$$M(v_0^{\phi_\psi}, y_0^{n-1}) = \underbrace{R(v_0^{\phi_\psi - 1} | y_0^{n-1}) + E(c, \mathbf{S})}_{R(v_0^{\phi_\psi} | y_0^{n-1})} - \Psi(\phi_\psi), \quad (5)$$

where $c = u_{\phi_{\psi-1}+1}^{\phi_\psi} A_{m_\psi}$ is a codeword of C_ψ.

B. Outer codes

The main idea of the proposed approach is to perform jointly the steps of the above described SDA, which correspond to the same block in the PD tree. Each combined step reduces to list decoding of the corresponding outer code $\mathcal{C}_\psi, \psi \in \mathcal{V}$.

Observe that the decoder of outer code $\mathcal{C}_\psi$ may produce at most 2^{k_ψ} codewords. Since we consider the sequential algorithm, one does not need to obtain these codewords immediately. Instead, the codewords of outer codes can be constructed one-by-one, i.e. once the corresponding path is extracted from the PQ. In this case these codewords should be constructed in the descending order of their ellipsoidal weight.

Moreover, in most cases it is sufficient to obtain just two such codewords, which can be computed in a simpler way compared to the full list decoding the outer code. Observe that this simplification is not possible in the context of SCL decoding of the polar code.

TABLE I: Variables used in BSDA

Variable	Description
l	index of a path $v_0^{\phi_{\psi_l}}$
q_i	Number of invocations of the i-th outer decoder
ψ_l	The index of outer decoder to be invoked for the l-th path
ϕ_i	The last phase of the i-th block
B_l	True if the l-th path should be cloned
m_j	$= \log_2 n_j$, where n_j is the length of outer code $\mathcal{C}_j$
R_l	Accumulated penalty $R(v_0^{\phi_{\psi_l}} \| y_0^{n-1})$ for the l-th path
$\tilde{R}_l$	$= R(v_0^{\phi_{\psi_l - 1}} \| y_0^{n-1})$
Z_l	Saved state for the last outer decoder used for the l-th path
M	Score of a path
$r(\psi)$	$\lfloor \phi_\psi / 2^{m_\psi} \rfloor$

We require that for each outer code $\mathcal{C}_\psi$ are available subroutines *Preprocess*$(\mathcal{C}_\psi, \mathbf{S}, Z)$ and *GetNextCodeword*$(\mathcal{C}_\psi, Z, \hat{c})$. The former performs some code-dependent preprocessing of LLR vector $\mathbf{S}$, and saves its results in a state variable Z. The latter uses Z to construct the next most probable codeword in the list, which is stored in the array given by pointer $\hat{c}$, and returns tuple $[e, b]$, where b is a boolean value, which is true iff more codewords can be obtained by the subsequent calls, and $e = E(\hat{c}, \mathbf{S})$. The structure Z includes the following fields:

- $\mathbf{S}$ — vector of LLRs.
- Any additional data needed for efficient recovery of codewords of $\mathcal{C}_\psi$ for given $\mathbf{S}$.

Note that the amount of codewords to be returned by *GetNextCodeword*$(\mathcal{C}_\psi, Z, \hat{c})$ is upper bounded by $\min(2^{k_\psi}, L)$.

C. The algorithm

Figure 2a illustrates the proposed block sequential decoding algorithm (BSDA). Table I presents the description of some of its internal variables. The input arguments for the algorithm are the LLRs $S_i = \log \frac{W(y_i|0)}{W(y_i|1)}$, where y_i is the result of transmission of codeword symbol c_i over a memoryless output-symmetric channel, maximal number of times L the decoder is allowed to pass via any phase or block, and maximal total number D of paths, which can be stored in the PQ.

Let us provide the brief description of the proposed algorithm. The algorithm makes use of the Tal-Vardy list decoder data structures [2]. The implementation based on the original ones is described in [34]. In this work we introduce some modifications, which are discussed in the Appendix. They avoid data copying and simplify the interface to outer decoders.

The algorithm starts from subroutine *Initialize* (see figure 2b), where the decoding data structures are initialized, input LLRs S_i are loaded and the initial path is pushed into the PQ.

The main loop of *BSDA* starts from extraction of path l with the best score M from the PQ. After that, the path clone operation is done (if it is possible) in line 10 in the *BackwardPass* function (see figure 2d). Then, the most probable continuation of the path l is constructed in the function *ForwardPass* (see figure 2c). After that, if $q_{\psi_l} \geq L$ then shorter paths are deleted from the PQ. Iterations are performed until a codeword is obtained and returned in line 7 of the *BSDA* function.

Below we discuss the algorithm in more details. We denote by ϕ_ψ the index of the last input symbol corresponding to the ψ-th outer $(n_\psi = 2^{m_\psi}, k_\psi)$ code $\mathcal{C}_\psi$, $\psi \in [\mathcal{V}]$. The partial

```
BSDA($S_0^{n-1}, L, D$)
1  INITIALIZE()
2  while $true$
3  do $(M, l) \leftarrow$ POPMAX()
4      if $r(\psi_l - 1)$ is odd
5         then ITERATIVELYUPDATEC($l, m - m_{\psi_l - 1}, r(\psi_{l-1})$)
6      if $\psi_l = \mathcal{V}$
7         then return GETARRAYPOINTERC_R($l, 0, 0$)
8      if $B_l = 1$
9         then REMOVEBADPATHS($D$)
10             BACKWARDPASS($l$) //path cloning
11     ITERATIVELYCALCS($l, m - m_{\psi_l}, r(\psi_l)$)
12     FORWARDPASS($l$) //extending the current path
13     $q_{\psi_l} \leftarrow q_{\psi_l} + 1$
14     if $q_{\psi_l} \geq L$
15        then for All paths $l'$ stored in the PQ
16              do if $\psi_l \leq \psi_{l'}$
17                    then KILLPATH($l'$)
18                         Remove $l'$ from the PQ
```

(a) The algorithm

```
INITIALIZE()
1  $l \leftarrow$ ASSIGNINITIALPATH()
2  PUSHPATH($0, l$)
3  $q_0^{\mathcal{V}-1} \leftarrow 0, \psi_l \leftarrow 0, R_l \leftarrow 0, B_l \leftarrow 0$
4  $s \leftarrow$ GETARRAYPOINTERS_W($l, 0$)
5  $s[i] \leftarrow S_i, 0 \leq i < n$
```

(b) Initialization of the algorithm

```
FORWARDPASS($l$)
1  $\mathbf{S} \leftarrow$ GETARRAYPOINTERS_W($l, m - m_{\psi_l}$)
2  $\hat{c} \leftarrow$ GETARRAYPOINTERC_W($l, m - m_{\psi_l}, r(\psi_l)$)
3  PREPROCESS($\mathcal{C}_{\psi_l}, \mathbf{S}, Z_l$)
4  $[e, b] \leftarrow$ GETNEXTCODEWORD($\mathcal{C}_{\psi_l}, Z_l, \hat{c}$)
5  $B_l \leftarrow b; \tilde{R}_l \leftarrow R_l; R_l \leftarrow R_l - e$
6  PUSH($R_l - \Psi(\phi_{\psi_l}), l$)
7  $\psi_l \leftarrow \psi_l + 1$
```

(c) Construction of the most probable codeword of outer code

```
BACKWARDPASS($l$)
1  $l' \leftarrow$ CLONEPATH($l$)
2  $\hat{c} \leftarrow$ GETARRAYPOINTERC_W($l', m - m_{\psi_{l-1}}, r(\psi_l - 1)$)
3  $[e, b] \leftarrow$ GETNEXTCODEWORD($\mathcal{C}_{\psi_l - 1}, Z_l, \hat{c}$)
4  $B_{l'} \leftarrow b; Z_{l'} \leftarrow Z_l; R_{l'} \leftarrow \tilde{R}_l - e; \tilde{R}_{l'} \leftarrow \tilde{R}_l; \psi_{l'} \leftarrow \psi_l$
5  PUSH($R_{l'} - \Psi(\phi_{\psi_l - 1}), l'$)
```

(d) On-demand construction of codewords of outer codes

Fig. 2: Block sequential decoding algorithm

sums of the input symbols v_i of the polarizing transformation, which are needed for computing of $S_m^{(i)}(v_0^{i-1}, y_0^{n-1})$, are updated in line 5, where $r(\psi) = \lfloor \phi_\psi / 2^{m_\psi} \rfloor$. Observe that in our algorithm we update the partial sums only for paths which were extracted from the PQ, while in case of list decoding this should be performed for each path in the list.

The boolean variable B_l is set to true iff at least one more codeword of code $\mathcal{C}_{\psi_l - 1}$ can be returned by the corresponding outer decoder. In this case the decoder ensures in line 9 (*RemoveBadPaths* procedure) that there are at most $D - 2$ entries in the PQ (if not, the paths with lowest scores are killed), and calls to *BackwardPass* function. This function constructs the next most probable codeword of $\mathcal{C}_{\psi_l - 1}$. This variable is set in the *ForwardPass* and *BackwardPass* functions.

In line 11 the vector of LLRs $\mathbf{S}$ is computed. The decoder makes a call to the *ForwardPass* algorithm, which constructs the most probable continuation of the l-th path, i.e. performs (near) maximum likelihood decoding of vector $\mathbf{S}$ in outer code. If the number of times q_{ψ_l} the decoder has visited the ψ_l-th block exceeds L, then paths shorter than ϕ_{ψ_l} are removed in line 18. The first steps of *ForwardPass* algorithm are to obtain writable pointers to the array $\mathbf{S}$ of log-likelihood ratios $S_{m-m_{\psi_l}}^{(r(\psi_l))}$, computed by *IterativelyCalcS*, and to the array $\hat{c}$, which is used to store the most probable continuation of the l-th path. In line 3 an appropriate pre-processing algorithm for $\mathcal{C}_{\psi_l}$ is invoked (see Section IV for details), and the most probable codeword is constructed in line 4. Variable e is assigned to the ellipsoidal weight of this codeword, while b is set to $true$ iff less probable codewords can be obtained by *GetNextCodeword* function. Finally, the value $R_l = R(v_0^{\phi_{\psi_l}}|y_0^{n-1})$, is updated according to Theorem 1, and the path is pushed to the priority queue. The previous value of R_l is saved in $\tilde{R}_l$, so that it can be used later to obtain the score of less probable continuations of this path.

The *BackwardPass* algorithm is used to obtain less probable codewords of outer codes in the descending order of their ellipsoidal weight. At line 1 the path is cloned. A writable pointer to the destination array for storing the codeword is obtained in line 2, and an appropriate codeword of the outer code is stored in this array.

The details of low-level functions *GetArrayPointer** used in the proposed algorithm are discussed in the Appendix.

Example 2. *Consider decoding of the* $(16, 10)$ *polar code* $\mathcal{C}$ *from the Example 1 in AWGN channel at* $E_b/N_0 = 5$ *dB.*

We need the values of bias function $\Psi(3) \approx -0.47, \Psi(7) \approx -0.52, \Psi(15) \approx -0.56$. *Let the input LLRs* $S_0^{(0)}$ *be equal to* $(0.44, 7.46, 7.19, 2.82, 5.63, 9.78, 6.06, -0.12, -0.64, 9.38, 10.87, 13.0, 13.43, 9.43, 2.02, 13.2)$. *Let* $l = 0$ *be the index of the initial path. At the first iteration, in line 11 the decoder computes the vector of LLRs* $S_2^{(0)}$, *which equals to* $(-0.44, 7.46, 2.02, -0.12)$. *ForwardPass function obtains codeword* $(1, 0, 0, 1) \in \mathcal{C}_0$ *with the ellipsoidal weight* $e = 0$. *Hence, in line 6 of the ForwardPass function a path with score* 0.47 *is pushed to the PQ.*

This path is extracted from the PQ at the next iteration of BSDA. BackwardPass function obtains codeword $(0, 0, 0, 0) \in \mathcal{C}_0$ *with* $e = -0.56$. *The path is cloned (let the ID of the cloned path be* $l' = 1$*), and an entry with score* $-0.56+0.47 = -0.09$ *is pushed to the PQ.*

The vector of LLRs $S_2^{(1)}$, *given by* $(6.08, 16.89, 9.2, -2.94)$, *is obtained at line 11 for path* 0. *Hence, one obtains codeword* $(0, 0, 0, 0) \in \mathcal{C}_1$ *with* $e = -2.94$ *by ForwardPass function, and path* 0 *is pushed to the PQ with score* $-2.94 + 0.52 = -2.42$.

At the next iteration of the decoder, path $l = 1$ *is extracted from the PQ. The vector of LLRs* $S_2^{(1)}$, *given by* $(5.19, 16.89, 9.2, 2.7)$, *is obtained in line 11. The codeword* $(0, 0, 0, 0) \in \mathcal{C}_1$ *is obtained with* $e = 0$, *and path* 1 *is pushed to the PQ with score* $-0.09 - 0 + 0.52 = 0.43$.

This path is extracted from the PQ at the next iteration. The LLRs $S_1^{(1)}$ *are equal to* $(-0.2, 16.84, 18.05, 15.82, 19.06, 19.2, 8.08, 13.08)$. *These values are preprocessed by the decoder for code* $\mathcal{C}_2$, *and the all-zero codeword with* $e = -0.2$ *is obtained in line 4 of the ForwardPass function. Hence, path* 1 *is pushed to the PQ with score* $-0.09 - 0.2 + 0.56 = 0.27$.

This path is extracted at the next iteration of the decoder, and, since all leaf nodes in the PD tree have been visited, the decoder terminates returning the all-zero codeword.

The proposed algorithm can be tailored to implement decoding of polar codes with CRC. To do this, one should add CRC validation to line 7 of the BSDA, so that iterations are performed until either a correct codeword is found, or no more paths remain in the PQ.

The proposed algorithm is not guaranteed to provide the same performance as the original SDA. In some cases its performance may be better, since the decoders for outer codes may avoid some errors of the sequential decoder. However, in some cases performance degradation may occur, if it happens that for an incorrect path v_0^{n-1} and some i $\forall j > i : M(v_0^{\phi_j}, y_0^{n-1}) > M(u_0^{\phi_i}, y_0^{n-1})$, and $\exists \tau \in (\phi_{j-1}, \phi_j] : \forall s \geq \phi_i \ M(v_0^{\tau}, y_0^{n-1}) < M(u_0^{s}, y_0^{n-1})$. That is, the proposed algorithm may miss the opportunity to switch to the correct path at an intermediate phase τ within some block, and proceed with exploration of an incorrect path. However, simulation results presented below show that the impact of this problem is negligible.

D. Hard decisions

In many cases the hard decision vector corresponding to some intermediate LLR vector $\mathbf{S}$ is error free, i.e. it is a codeword of $\mathcal{C}_i$. In this case one should avoid invoking a relatively complex soft-decision decoding algorithm of outer code, i.e. *Preprocess* and *GetNextCodeword* functions in lines 3-4 of *ForwardPass*, unless non-ML codewords of the corresponding outer code are needed.

Consider some decoding iteration and suppose that path l is extracted from the PQ. Let us construct the hard decision vector $\bar{c}$ of $\mathbf{S}$. If $\bar{c} \in \mathcal{C}_i$, then we can immediately set $e \leftarrow 0$, $b \leftarrow 1$ and push the path to the PQ. The LLR vector $\mathbf{S}$ is saved in the state variable Z_l, so that computationally expensive preprocessing can be done later.

If the hard decision vector is a valid codeword of the corresponding outer code, then it is very likely that the less probable codewords will not be needed during the next iterations of the decoding. Hence, it is possible to skip construction of such codewords. However, occasionally such codewords may be needed, and some provision needs to be done in order to recover them later. It can be easily seen that the ellipsoidal weight of any such codeword cannot be more than $-d_{\psi_l - 1} \min_i |Z_l.\mathbf{S}_i|$, where $d_{\psi_l - 1}$ is the minimum distance of $\mathcal{C}_{\psi_l - 1}$. We propose to use this value for computing an estimate of $R_{l'}$ of the less probable path l'. If this path is later selected by the decoder for further processing, the corresponding codeword should be actually constructed.

IV. Decoding of outer codes

As described in Section III-A, PD is applied recursively until one obtains outer codes, which allow efficient ML decoding. Consider some (n, k) outer code $\mathcal{C}$. We need to construct a decoder, which can find the codewords $c^{(i)} \in \mathcal{C}$ in the increasing order of their ellipsoidal weight $E(c^{(i)}, S_0^{n-1})$, where S_0^{n-1} is the vector of LLRs. In [34] outer codes are decoded with tree-trellis Viterbi algorithm. However, in many cases it is possible to use much simpler algorithms.

In this section we describe the decoding algorithms for outer codes, which frequently arise in PD of polar codes. Some of the techniques presented below resemble those suggested in [7], but we also consider some well-known outer codes, most importantly first-order Reed-Muller and extended Hamming codes.

A. Low rate codes

Decoding of $(n, 0), (n, 1)$ and $(n, 2)$ codes is performed by exhaustive enumeration of their codewords $c^{(i)}$, computing the corresponding ellipsoidal weight $E(c^{(i)}, S_0^{n-1})$ for each codeword, and sorting them in the ascending order of $E(c^{(i)}, S_0^{n-1})$.

B. First order Reed-Muller and related codes

The first order Reed-Muller code $RM(1, \mu)$ is obtained as a polar code with the frozen set $\hat{\mathcal{F}} = [2^{\mu}] \setminus \left(\{0\} \cup \left\{2^i | 0 \leq i < \mu\right\}\right)$. List decoding of such codes can be implemented using the fast Hadamard transform (FHT) with complexity $O(n \log n)$ [35]. FHT computes correlations $T(c^{(i)}, S_0^{n-1}) = \sum_{j=0}^{n-1} (-1)^{c_j^{(i)}} S_j$ for n codewords of the corresponding codes. The correlations for the remaining codewords are given by $T(c^{(i+n)}, S_0^{n-1}) = -T(c^{(i)}, S_0^{n-1})$, and $c^{(i+n)} = c^{(i)} + \mathbf{1}$, where $\mathbf{1}$ is a vector of 1's. The ellipsoidal weight of a codeword is related to its correlation by $E(c^{(i)}, S_0^{n-1}) = \frac{1}{2}\left(\sum_{j=0}^{n-1} |S_j| - T(c^{(i)}, S_0^{n-1})\right)$.

Observe that obtaining two most probable codewords, which are in most cases sufficient for the BSDA, requires finding just two highest values $T(c^{(i)}, S_0^{n-1})$.

Another type of outer codes, commonly arising in the PD of polar codes, is a concatenation of a first order Reed-Muller code $RM(1, \mu - t)$ and a $(2^t, 1, 2^t)$ repetition code. Such codes may be also decoded using the FHT of order $2^{\mu - t}$. We propose also to use FHT-based decoder for the case of codes given by a union of at most 4 cosets of a first order Reed-Muller code $\mathcal{R}$, i.e. $\mathcal{C} = \mathcal{R} \cup (\mathcal{R} + c')$, and $\mathcal{C} = \mathcal{R} \cup (\mathcal{R} + c') \cup (\mathcal{R} + c'') \cup (\mathcal{R} + c' + c'')$, where $c', c'' \notin \mathcal{R}$. This turns out to be more efficient in practice than performing additional steps of PD.

C. Single parity check code

We perform decoding of $(n, n-1, 2)$ codes by testing a few pre-defined error patterns $\mathcal{E}^{(i)}$. This method is known as Chase-II decoding algorithm [36] and was used in [7]. First, the codeword symbols are arranged in the increasing order of their reliabilities, so that $|S_{t[0]}| \leq |S_{t[1]}| \leq \ldots |S_{t[n-1]}|$. Second, a hard decision vector $\hat{c}$ is constructed, and its parity p is calculated. Then the codewords are constructed as $c^{(i)} = \hat{c} + e^{(i)}$, where $e^{(i)}$ is the vector containing 1's on positions $t[\epsilon_{i,j}]$ and 0's elsewhere, for all $\mathcal{E}^{(i)} = \{\epsilon_{i,0}, \ldots, \epsilon_{i,w_i}\} \in \mathbf{T}^{(p)}$. The set of test error patterns $\mathbf{T}^{(p)}$ can be constructed either analytically using the expressions derived in [37], or by simulations. Table II presents the test error patterns used in BSDA. These patterns were obtained via simulations. It turns out that the same set of test error patterns can be used for decoding of codes of arbitrary length without any noticeable performance loss compared with the optimal decoder.

TABLE II: Test error patterns for single parity check code

$\mathbf{T}^{(1)}$	$\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{0,1,2\}$, $\{0,1,3\}$, $\{0,2,3\}$, $\{1,2,3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{7\}$, $\{0,1,4\}$, $\{0,1,5\}$, $\{0,1,6\}$, $\{0,2,4\}$, $\{0,3,4\}$, $\{8\}$, $\{9\}$, $\{10\}$, $\{11\}$, $\{12\}$
$\mathbf{T}^{(0)}$	$\{\}$,$\{0,1\}$, $\{0,2\}$, $\{0,3\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{0,1,2,3\}$, $\{0,4\}$, $\{0,5\}$, $\{0,6\}$, $\{0,7\}$, $\{1,4\}$, $\{1,5\}$, $\{1,6\}$, $\{1,7\}$, $\{2,4\}$, $\{2,5\}$, $\{2,6\}$, $\{3,4\}$, $\{3,5\}$, $\{0,1,2,4\}$, $\{0,8\}$, $\{0,9\}$ $\{0,10\}$, $\{0,11\}$

The Chase-II decoding may result in performance degradation of BSDA, since the considered algorithm does not necessarily return true L most reliable codewords. In this case one should increase the size of $\mathbf{T}^{(p)}$ and/or reduce the maximal allowed length of single parity check (SPC) code. Observe that the fast implementation of SCL in [7] uses only 8 test error patterns. In our implementation we use 26 ones. Furthermore, simulations show, that in most cases it is sufficient to identify the positions $t[0], t[1]$ of only two least reliable symbols. This can be done using the tournament algorithm [38].

D. Double parity check codes

A $(n, n-2, 2)$ polar code with the set of frozen symbol indices $\mathcal{F} = \{0, 1\}$ can be obtained by interleaving two $(n/2, n/2 - 1, 2)$ codes. This enables one to decode such codes using a combination of two decoders of a SPC code.

E. Rate-1 code

For (n, n) codes we propose to use the same decoding algorithm as for SPC codes (see Section IV-C). Moreover, simulations show that finding just 4 (out of 2^n) most probable codewords of (n, n) code does not result in any noticeable performance loss for considered error rates. These 4 most probable codewords is obtained by considering the following error patterns: $\emptyset, \{0\}, \{1\}, \{0, 1\}, \{2\}$. Their computation requires identification only 3 smallest values $|S_j|, j \in [n]$.

F. $(16, 10, 4)$, $(16, 11, 4)$ and $(16, 12, 2)$ codes

These codes, obtained by Plotkin concatenation of $(8, 4, 4)$ or $(8, 3, 4)$ codes and $(8, 7, 2)$ or $(8, 8, 1)$ codes, commonly arise in the PD of polar codes. Decoding of these codes can be implemented using the approach introduced in [39].

V. Block sequential decoding for polar subcodes

A. Dynamic frozen symbols

It was suggested in [3] to set frozen symbols $u_i, i \in \mathcal{F}$ not to zero, but to linear combinations of some other symbols, i.e.

$$u_i = \sum_{s=0}^{i-1} V_{j_i, s} u_s, \tag{6}$$

```
PrepareForDFEvaluation($l, \hat{c}$)
1  for $j \in \mathcal{P} \cap \{j \mid \phi_{\psi_l} - 2^{m_{\psi_l}} < j' \le \phi_{\psi_l}\}$
2  do $v_j \leftarrow (\hat{c}A_{m_{\psi_l}})_{j \bmod 2^{m_{\psi_l}}}$
3     if $v_j = 1$
4       then for $i \leftarrow 0$ to $f-1$
5              do $w_{l,i} \leftarrow w_{l,i} + V_{i,j}$
```

Fig. 3: Accumulating the values of dynamic frozen symbols

where V is a $(n-k)\times n$ binary matrix, such that its rows end in distinct columns, and j_i is the index of row with the last non-zero element in column i. Such symbols with non-trivial right hand side expressions are called dynamic frozen symbols (DFS), and the code obtained via considered construction are referred to as *polar subcodes*. Decoding of such codes can be implemented by a straightforward generalization of the successive cancellation algorithm and SC-based algorithms.

Properly constructed polar subcodes may have higher minimum distance than classical polar codes. This results in substantially better performance [3], [40] under the SCL algorithm and other SC-based algorithms. Polar codes with CRC [2] can be considered as a special case of polar subcodes.

A system of dynamic freezing constraints (DFC) may be constructed for any linear code of length 2^m with check matrix H, by setting $V = QHA_m^T$, where Q is a suitable invertible matrix. This enables one to decode such code with the SCL algorithm [3].

B. Processing of dynamic frozen symbols

Decoding of polar subcodes requires one to compute the values of DFS, i.e. some linear combinations of symbols v_i for any path $v_0^{\phi_{\psi_l}}$. The Tal-Vardy list decoding algorithm does not store these values explicitly. It is possible to express their values from the content of arrays $C_{l,\lambda}$. However, we employ an alternative approach, which is more efficient in practice.

In most cases, polar subcodes have only a few non-trivial DFS which depend on a small number of other symbols. Let f be the number of non-trivial equations (6) for the considered code. It can be assumed without loss of generality that these equations correspond to f topmost rows of matrix V. Let $i_s \in \mathcal{F}, 0 \le s < f$, be the indices of the corresponding dynamic frozen symbols. Let $\mathcal{P} = \{j \mid V_{s,j} = 1, 0 \le j < i_s, 0 \le s < f\}$ be the set of indices of symbols participating in any of the DFC.

We propose to allocate boolean variables $w_{l,s}, 0 \le l < D$ for each path, initialize them to 0 at decoder startup, and flip the value of $w_{l,s}$ at each phase $j < i_s$, such that $V_{s,j} = 1$ and $v_j = 1$, where v_j is the value of the j-th symbol on the l-th path. Then at phase i_s the value of $w_{l,s}$ is exactly the value of the s-th DFS for the corresponding path. However, the above described BSDA does not compute explicitly the values v_j. But one can obtain these values as $v_j = (\hat{c}A_{m_{\psi_l}})_{j \bmod 2^{m_{\psi_l}}}$, where $\hat{c}$ is a codeword of an outer code obtained for path l at block ψ_l. This approach is illustrated in Figure 3. Observe that lines 2 and 4–5 of the algorithm can be efficiently implemented via bit mask manipulation techniques.

If there is a non-trivial DFS in some block ψ_l, i.e. $\phi_{\psi_l} - 2^{m_{\psi_l}} < i_s \le \phi_{\psi_l}$ for some s, and $v_{l,s} = 1$ when the decoder reaches this block, then one should perform decoding in a non-trivial coset of the corresponding outer code. The coset representative is given by

$$\mathbf{p}_s = \sum_{\substack{i=\phi_{\psi_l}-2^{m_{\psi_l}}+1 \\ V_{s,i}=1}}^{\phi_{\psi_l}} (A_{m_{\psi_l}})_{i \bmod 2^{m_{\psi_l}},-}, \qquad (7)$$

where $B_{i,-}$ denotes the i-th row of matrix B.

We introduce the algorithm *GetCoset*, which computes the value

$$\mathbf{p} = \sum_{s \in \mathfrak{S}} w_{l,s}\mathbf{p}_s,$$

where $\mathfrak{S} = \{s \mid \phi_{\psi_l} - 2^{m_{\psi_l}} < i_s \le \phi_{\psi_l}\}$. During the decoding process, *GetCoset* should be called before the line 3 of the *ForwardPass* function (Figure 2c) and adjust the signs of the LLRs $\mathbf{S}$, i.e. $\mathbf{S}_i \leftarrow \mathbf{S}_i(-1)^{\mathbf{p}_i}$. After that, each codeword $\hat{c}$ returned from the outer decoder (including *BackwardPass* function) should be corrected according to $\mathbf{p}$, i.e. $\hat{c} \leftarrow \hat{c} + \mathbf{p}$. Note that the vectors $\mathbf{p}_s$ can be precomputed offline. The corrected vector $\hat{c}$ should be passed to *PrepareForDFEvaluation* function to update values $w_{l,s}$.

VI. Complexity analysis

Consider block sequential decoding of $(n = 2^m, k)$ polar code. Let $\mathcal{V}$ be a number of outer codes. The worst-case complexity of the proposed decoding algorithm corresponds to the case when exactly $L\mathcal{V}$ iterations are performed, i.e. $q_\psi = L, 0 \le \psi < \mathcal{V}$. In this case the number of operations performed by the decoder is given by

$$C \le L \sum_{\psi=0}^{\mathcal{V}-1} \left(C'_\psi + C''_{\psi-1} + \Lambda(m - m_\psi, \lfloor \phi_\psi / 2^{m_\psi} \rfloor)\right), \qquad (8)$$

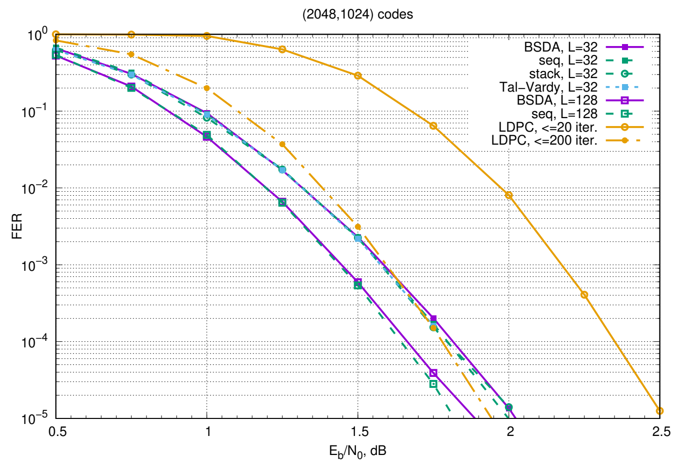
where C'_ψ is the complexity of a call to *Preprocess* and *GetNextCodeword* (see *ForwardPass* function) for outer code $\mathcal{C}_\psi$, C''_ψ is the complexity of subsequent calls[3] to *GetNextCodeword* (see *BackwardPass*). Here $\Lambda(\lambda, \phi) = 2^{m-\lambda}(2^{d+1} - 1)$ is the complexity of computing $S_\lambda^{(\phi)}$ via function *IterativelyCalcS*, where $d = d(\phi) < \lambda$ is the maximal integer, such that $2^{d(\phi)} | \phi$.

Application of the proposed approach makes sense only if *Preprocess* and *GetNextCodeword* functions provide a simpler way to obtain L most probable codewords of $\mathcal{C}_\psi$ compared with the Tal-Vardy algorithm[4] with list size L. Hence, the worst-case complexity of the proposed approach can be upper-bounded by considering the case (this corresponds to the algorithm presented in [14]) of $m_i = 0$. In this case one has $C'_\psi = C''_\psi = 0, 0 \le \psi < \mathcal{V} = n$, and

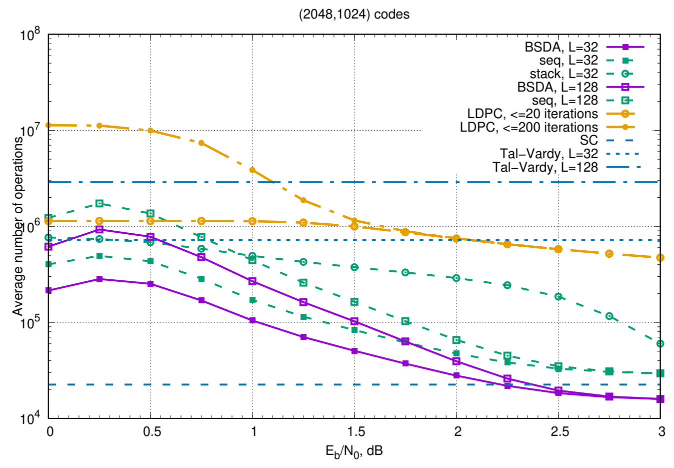
$$C \le L \sum_{\phi=0}^{n-1} \Lambda(m, \phi) = \sum_{\phi=0}^{n-1} (2^{d(\phi)+1} - 1).$$

[3]We assume $C''_{-1} = 0$.

[4]L must be sufficiently large to ensure that the Tal-Vardy algorithm always finds L most probable codewords.



(a) Performance BSDA



(b) Complexity

Fig. 4: Performance and complexity of decoding algorithms for $(2048, 1024)$ codes

For any $d < m-1$ there are 2^{m-d-1} integers $\phi < 2^m$ divisible by 2^d (and 2 of them for $d = m-1$), but not divisible by 2^{d+1}. Hence, one obtains

$$C \leq L(2^m - 1 + \sum_{d=0}^{m-1} 2^{m-d-1}(2^{d+1}-1)) = Lm2^m,$$

which is identical to the complexity of the SCL decoding. The best case complexity corresponds to the case when the decoder visits each block once, so it is given by $n \log_2 n$ with $L = 1$.

There are additional costs associated with PQ operations. With appropriate implementation [25], [26], their complexity is upper bounded by $O(D\mathcal{V})$.

VII. Numeric results

Figure 4a illustrates the performance of the proposed BSDA. Simulations were run for the case of AWGN channel, BPSK modulation and randomized polar subcode [4]. For comparison, we report also the performance of list [2], sequential [15] and min-sum stack [27] decoding algorithms for the same code, and the CCSDS LDPC code under belief propagation decoding. It can be seen that the proposed algorithm provides essentially the same performance as the sequential and Tal-Vardy algorithms. Furthermore, for $L = 32$ its performance is close to that of the LDPC code with at most 200 decoder iterations. Even better performance is obtained for $L = 128$.

Figure 4b illustrates the average number of summation and comparison operations performed by the considered algorithms. It can be seen that the complexity of the SDA is much lower compared with the original stack algorithm (which corresponds to $\Psi(\phi) = 0, 0 \leq \phi < n$). Furthermore, the average complexity of the block sequential algorithm converges quickly to a value slightly less than $n \log_2 n$, the complexity of the SC algorithm. The complexity of the proposed algorithm is 1.5–2 times lower compared with that of the sequential decoder, and substantially lower compared with $Ln \log_2 n$, the complexity of the Tal-Vardy list decoding algorithm, and the average complexity of the min-sum stack decoding algorithm.

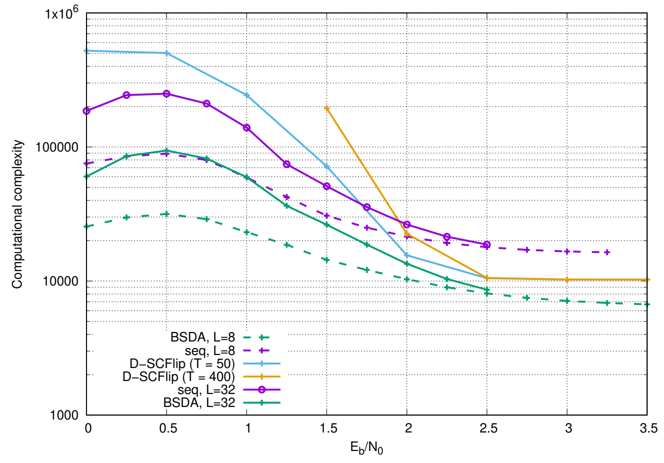
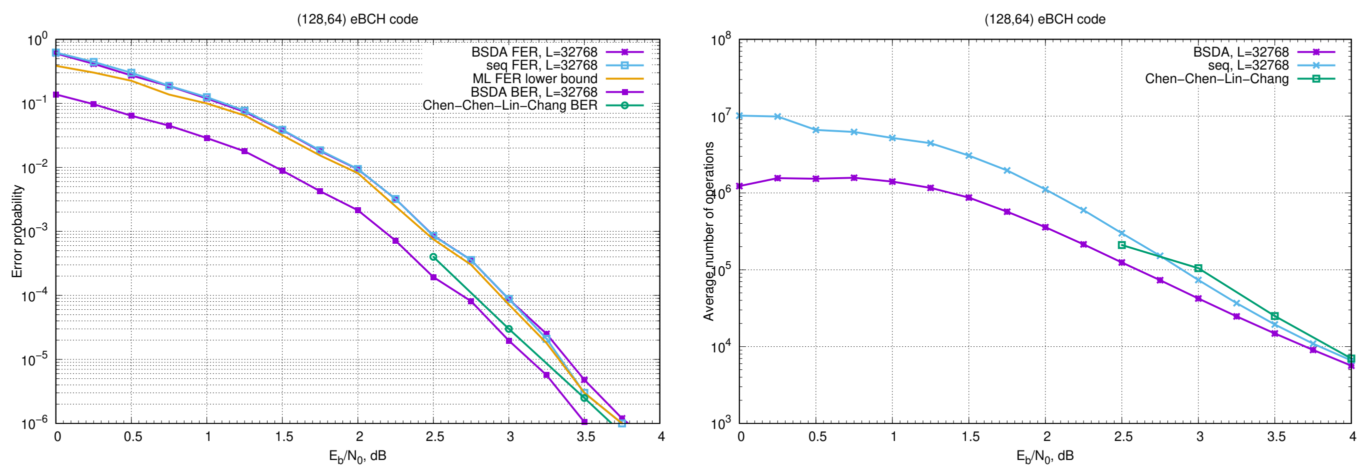


Fig. 5: Complexity of decoding algorithms, (1024,512) code

It is also substantially lower compared with the complexity of the BP decoder for the LDPC code. Observe that reducing the maximal number of iterations for the BP algorithm results in a noticeable performance degradation without significant complexity reduction for $FER < 0.1$.

Figure 5 presents the average complexity of BSDA, sequential and SC Flip [41] decoding algorithms. It can be seen that the complexity of the sequential decoder with $L = 8$ becomes higher than D-SCFlip with parameter $T = 50$, which corresponds to $L = 8$ in SCL. On the contrary, BSDA has lower decoding complexity even with $L = 32$.

As mentioned in Section V, any binary linear block code can be decoded with the proposed algorithm, although the performance of such a decoder depends strongly on the structure of the corresponding frozen set. eBCH codes were shown to have sufficiently low SC decoding error probability [3], and are therefore well-suited for decoding using the BSDA. Figure 6 illustrates performance and complexity of the decoding of $(128, 64, 22)$ eBCH code. For comparison, we report also the results for Chen-Chen-Lin-Chang algorithm (a sequential-type



(a) Performance

(b) Complexity

Fig. 6: Block sequential decoding of $(128, 64)$ eBCH code

trellis-based decoding method), reproduced from [24]. It can be seen that the BSDA provides lower decoding complexity.

Figure 7 illustrates the performance and throughput of the software implementation of the proposed BSDA, as well as fast list and adaptive list (ASCL) decoding algorithms introduced in [7], for the case of polar subcodes and polar codes with CRC-8. Simulations were performed on Intel Core i7-2600K CPU running at 3.4 GHz with maximum turbo frequency 3.8 GHz. SIMD techniques introduced in [42], [7], based on single-precision floating point arithmetic, were used to implement LLR computation in the proposed algorithm. Throughput results for the fast and ASCL decoding algorithms are reproduced from [7]. The performance of polar codes with CRC under the BSDA is very close to that of the list decoder with the same L, and is therefore not shown. As it may be expected, polar subcodes provide better performance than polar codes with CRC, and increasing list size L results in better performance. One can see that, for polar subcodes, at sufficiently high SNR the proposed BSDA even for $L = 32$ provides the same or even better average throughput as the fast list decoding algorithm introduced in [7] for polar codes with CRC and $L = 2$. Furthermore, at high SNR the throughput of BSDA for polar codes with CRC exceeds that of the fast list decoding algorithm. Observe that the algorithm presented in [7] relies on unrolling to eliminate redundant calculations, i.e. the decoder is specific for each code. The proposed BSDA is generic, but still provides higher throughput despite of much more sophisticated flow control structure.

It can be also seen that for a $(2048, 1723)$ polar subcode and $E_b/N_0 < 4.2$ dB the BSDA provides higher throughput and substantially better performance compared with the ASCL decoding algorithm [7] for a polar code with CRC-32. However, for higher values of E_b/N_0 the throughput of the ASCL decoding becomes much higher. The reason for this is that in this case with high probability the decoding is successful already with $L = 1$ (i.e. with plain SC decoding), and this can be easily verified by CRC. Hence, highly complex list decoder is almost not used. It is, however, not clear how to extend the idea of adaptive list decoding to the case of polar subcodes, which provide much better performance.

TABLE III: Decoder memory requirements

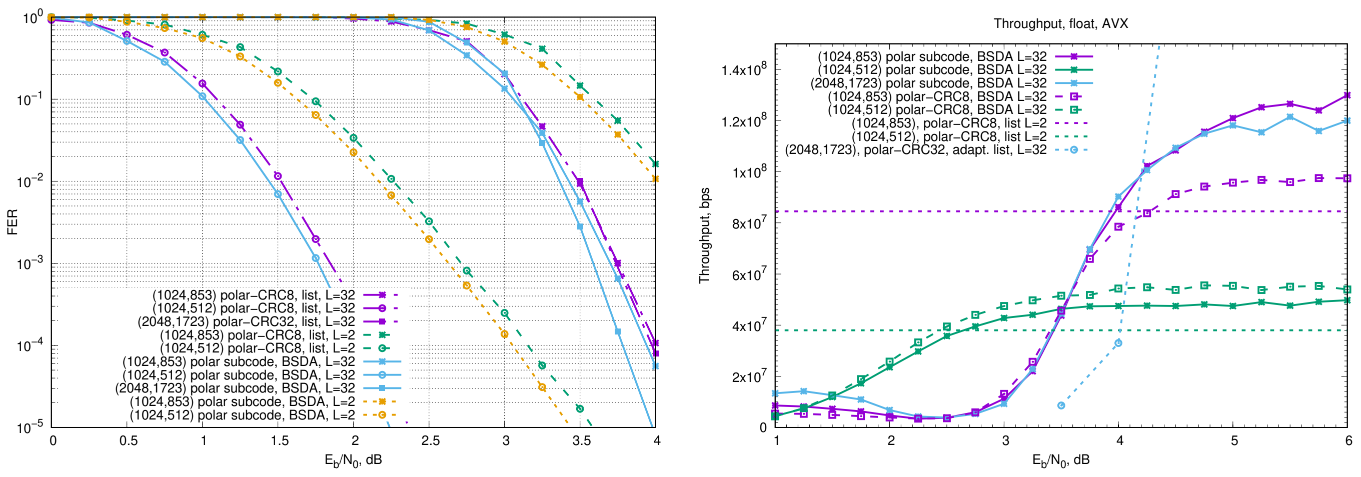
L	D	Ξ	Ξ_{TV}	L	D	Ξ, KB	Ξ_{TV}
(1024, 512, 28)				(16384, 8192, 48)			
8	70	385	49	8	250	4764	786
32	240	1457	197	32	900	18617	3146
256	1620	11254	1572	256	8230	160791	25165
(2048, 1024, 48)				(2048, 683, 52)			
8	100	786	98	8	100	681	98
32	370	3071	393	32	400	2686	393
256	3020	24215	3146	256	2450	20859	3146

Table III presents the amount of memory used by the decoder in various scenarios. The value of Ξ is the maximal amount of memory (in Kilobytes) sufficient for storing arrays S, C, and outer decoder state variables Z from the common memory pools, described in Appendix B. The values of parameters L, D were selected to minimize overall memory demand during block sequential decoding, while ensuring that the performance does not degrade with respect to the case of $D = Lk$, which corresponds to the maximal possible memory footprint. Minimization for each code was carried out for FER at 10^{-3}. Observe that for an SCL decoder one needs to store Ln LLRs S and $2Ln$ partial sums C. For a software implementation, this results in $6Ln$ bytes of storage. The corresponding values are shown as Ξ_{TV} in the table. It can be seen that Ξ/Ξ_{TV} decreases with code length.

VIII. Conclusions

In this paper the block sequential decoding algorithm was introduced. It employs blockwise processing of the input symbols of the polarizing transformation. The processing operation reduces to on-demand construction of codewords of the codes arising in the Plotkin decomposition of the code being decoded. A set of such codes was identified, which admits low complexity list decoding.

It was shown that the proposed block sequential decoding algorithm has lower complexity than the sequential, stack



(a) Performance

(b) Throughput of the software implementation

Fig. 7: Performance and throughput of block sequential and fast list decoding algorithms

and list decoding algorithms, while having approximately the same performance. At sufficiently high SNR, the throughput of the software implementation of the proposed algorithm exceeds the throughput of the fast list decoder with much smaller list size, i.e. the proposed algorithm provides better performance and lower decoding complexity compared with the list decoding algorithm by Sarkis et al [7]. The proposed algorithm can be used for decoding of polar (sub)codes, polar codes with CRC and short eBCH codes.

Appendix

A. Data structures and basic procedures

The proposed decoding algorithm can be implemented using the techniques suggested in [2]. However, several simplifications are possible. Let l, λ, ϕ, β denote the path, layer, phase and branch number, respectively. Each path is associated with arrays of intermediate LLRs $S_{l,\lambda}[\beta], 0 \leq l < D, 0 \leq \lambda \leq m-\mu, 0 \leq \beta < 2^{m-\lambda}$, where D is the maximal number of paths considered by the decoder (i.e. the maximal size of the PQ), and $2^\mu, 0 \leq \mu < m$, is the length of the shortest outer code in the Plotkin decomposition tree, i.e. $\mu = \min_{\psi \in [\mathcal{V}]} m_\psi$. Each path is also associated with value R_l, which contains values $R(u_0^\phi|y_0^{n-1})$, similarly to [29], [43].

It was suggested in [2] to store the arrays of partial sum tuples $C_{l,\lambda}[\beta][\phi \bmod 2]$. We propose to rename these arrays to $C_{l,\lambda,\phi \bmod 2}[\beta]$. By examining the *RecursivelyUpdateC* algorithm presented in [2], one can see that $C_{l,\lambda,1}[\beta]$ is just copied to $C_{l,\lambda-1,\psi}[2\beta+1]$ for some $\psi \in \{0,1\}$, and this copy operation terminates on some layer λ'. Observe that $\lambda - \lambda'$ is equal to the maximal integer d, such that $\phi+1$ is divisible by 2^d. Therefore, we propose to co-locate $C_{l,\lambda,1}[\beta]$ with $C_{l,\lambda_0,0}[\beta]$. In this case the corresponding pointers are given by $C_{l,\lambda,1} = C_{l,\lambda_0,0} + 2^{m-\lambda}(2^{\lambda-\lambda'}-1)$. This not only results in the reduction of the amount of data stored by a factor of two, but also enables one to avoid "copy on write" operation (see line 6 of Algorithm 9 in [2]). Therefore, we write C_{l,λ_0} instead of $C_{l,\lambda_0,0}$ in what follows.

We use the array pointer mechanism suggested in [2] to avoid data copying. However, we distinguish the case of read and write data access. Retrieving read-only pointers is performed by functions *GetArrayPointerC_R*(l,λ) and *GetArrayPointerS_R*(l,λ) shown in Figure 10. Retrieving writable pointers is performed by function *GetArrayPointerW*(T,l,λ), where $T \in \{'C','S'\}$ shown in Figure 9. This function implements reference counting mechanism similar to that proposed in [2]. It is discussed in more details in Section B.

Figures 8a and 8b present iterative algorithms for computing $S_{l,\lambda}[\beta]$ and $C_{l,\lambda}[\beta]$. These algorithms resemble the recursive ones given in [2]. However, the proposed implementation avoids costly array dereferencing operations.

B. Memory management

Many paths considered by the proposed algorithm share common values of $S_{l,\lambda}[\beta]$ and $C_{l,\lambda}[\beta]$, similarly to SCL decoding. To avoid duplicate calculations one can use the same shared memory data structures. That is, for each path l and for each layer λ we store the index of the array containing the corresponding values $S_{l,\lambda}[\beta]$ and $C_{l,\lambda}[\beta]$. This index is given by $p =$*PathIndex2ArrayIndex*$[l,\lambda]$, so that the corresponding data can be accessed as *ArrayPointer*$[T][p], T \in \{'S','C'\}$. Furthermore, for each integer p we maintain the number of references to this array *ArrayReferenceCount*$[p]$. If the decoder needs to write the data into an array, which is referenced by more than one path, a new array needs to be allocated. Observe that there is no need to copy anything into this array, since it will be immediately overwritten. This is an important advantage with respect to the implementation described in [2]. However, the sequence of array read/write and stack push/pop operations still satisfies the validity assumptions introduced in [2], so the proposed algorithm can be shown to be well-defined by exactly the same reasoning as the original SCL.

Only one path considered by the decoder is constructed to the full length n. For most of the paths, only a few symbols are constructed before these paths are abandoned, i.e. either stored without being accessed till the decoder terminates, or killed.

```
ITERATIVELYCALCS($l, \lambda, \phi$)
  $d \leftarrow \max\left\{0 \le d' \le \lambda - 1 | \phi \text{ is divisible by } 2^{d'}\right\}$
  $\lambda' \leftarrow \lambda - d$
  $S' \leftarrow$ GETARRAYPOINTERS_R($l, \lambda' - 1$)
  $N \leftarrow 2^{m-\lambda'}$
  if $\phi 2^{-d}$ is odd
    then $\tilde{C} \leftarrow$ GETARRAYPOINTERC_R($l, \lambda'$)
         $S'' \leftarrow$ GETARRAYPOINTERS_W($l, \lambda'$)
         $S''[\beta] \leftarrow$ P($\tilde{C}[\beta], S'[\beta], S'[\beta + N]$), $0 \le \beta < N$
         $S' \leftarrow S''; \lambda' \leftarrow \lambda' + 1; N \leftarrow N/2$
  while $\lambda' \le \lambda$
  do $S'' \leftarrow$ GETARRAYPOINTERS_W($l, \lambda'$)
     $S''[\beta] \leftarrow$ Q($S'[\beta + N], S'[\beta]$), $0 \le \beta < N$
     $S' \leftarrow S''; \lambda' \leftarrow \lambda' + 1; N \leftarrow N/2$
```

(a) Computing $S_\lambda^{(\phi)}(u_0^{\phi-1}, y_0^{N-1})$

```
ITERATIVELYUPDATEC($l, \lambda, \phi$)
  $\delta \leftarrow \max\left\{d | \phi + 1 \text{ is divisible by } 2^d\right\}$
  $\tilde{C} \leftarrow$ GETARRAYPOINTERC_W($l, \lambda - \delta, 0$)
  $N \leftarrow 2^{m-\lambda}; \tilde{C} = \tilde{C} + N(2^\delta - 2); C'' \leftarrow \tilde{C} + N;$
  $\lambda' \leftarrow \lambda - \delta$
  while $\lambda > \lambda'$
  do $C' \leftarrow$ GETARRAYPOINTERC_R($l, \lambda$)
     $\tilde{C}[\beta] \leftarrow C'[\beta] \oplus C''[\beta], 0 \le \beta < N$
     $N \leftarrow 2N; C'' \leftarrow \tilde{C}; \tilde{C} \leftarrow \tilde{C} - N$
     $\lambda \leftarrow \lambda - 1$
```

(b) Updating C arrays

Fig. 8: Computing LLRs and partial sums

```
GETARRAYPOINTERW($T, l, \lambda$)
  $p \leftarrow PathIndex2ArrayIndex[l, \lambda]$
  if $p = -1$
    then $p \leftarrow$ ALLOCATE($\lambda$)
    else if $ArrayReferenceCount[p] > 1$
           then $ArrayReferenceCount[p] - -$
                $p \leftarrow$ ALLOCATE($\lambda$)
  return $ArrayPointer[T][p]$

GETARRAYPOINTERS_W($l, \lambda$)
  return GETARRAYPOINTERW($'S', l, \lambda$)

GETARRAYPOINTERC_W($l, \lambda, \phi$)
  $\delta \leftarrow \max\left\{d | \phi + 1 \text{ is divisible by } 2^d\right\}$
  $C \leftarrow$ GETARRAYPOINTERW($'C', l, \lambda - \delta$)
  if $\phi \equiv 1 \bmod 2$
    then $C \leftarrow C + 2^{m-\lambda}(2^\delta - 1)$
  return $C$
```

Fig. 9: Write access to the data

```
GETARRAYPOINTERS_R($l, \lambda$)
  return $ArrayPointer['S'][PathIndex2ArrayIndex[l, \lambda]]$

GETARRAYPOINTERC_R($l, \lambda$)
  return $ArrayPointer['C'][PathIndex2ArrayIndex[l, \lambda]]$
```

Fig. 10: Read-only access to the data

```
ALLOCATE($\lambda$)
  $[t, q] \leftarrow Pop(InactiveArrayIndices[\lambda]);$
  $t \leftarrow t(m + 1) + \lambda; ArrayReferenceCount[t] = 1$
  if $q = 1$
    then if $\Phi > \Lambda$
           then ABORT
         $ArrayPointer['S'][t] = PoolS + \Phi;$
         $ArrayPointer['C'][t] = PoolC + \Phi$
         $\Phi += 2^{m-\lambda}$
  return $t$
```

Fig. 11: Adaptive memory allocation

Hence, one does not need to provide the memory needed to accommodate all D paths. Therefore, we propose to create common memory pools for arrays C and S, denoted *PoolC* and *PoolS*, respectively. If a new array needs to be provisioned, a part of memory pool is assigned to it. Arrays C and S are provisioned simultaneously. Let Φ denote the amount of memory consumed from these pools. If Φ exceeds the size of the memory pools Λ, then decoding needs to be terminated. For sufficiently large Λ this typically occurs after the correct path has been killed by the decoder.

The stack *InactiveArrayIndices* stores pairs $[t, q]$, where t is an index of an array, and q is true if the array is not allocated yet. If the number of references to some array drops to 0, then the index of the array is saved in a stack of unused arrays, similarly to [2], so that it can be re-used later. The indices of unused arrays corresponding to different layers are stored in different stacks *InactiveArrayIndices*$[\lambda]$, since these arrays have different sizes. Allocation from the common pools occurs only if the corresponding index is extracted for the first time ($q = 1$ in Figure 11, which illustrates the proposed approach).

REFERENCES

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

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

[3] P. Trifonov and V. Miloslavskaya, “Polar subcodes,” *IEEE J. Sel. Areas Commun.*, vol. 34, no. 2, pp. 254–266, February 2016.

[4] P. Trifonov and G. Trofimiuk, “A randomized construction of polar subcodes,” in *Proceedings of IEEE ISIT*, 2017, pp. 1863–1867.

[5] A. Alamdar-Yazdi and F. Kschischang, “A simplified successive-cancellation decoder for polar codes,” *IEEE Commun. Lett.*, vol. 15, no. 12, December 2011.

[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] ——, “Fast list decoders for polar codes,” *IEEE Journal On Selected Areas In Communications*, vol. 34, no. 2, pp. 318–328, February 2016.

[8] S. A. Hashemi, C. Condo, and W. Gross, “Fast and flexible successive-cancellation list decoders for polar codes,” *IEEE Trans. Signal Process.*, vol. 65, no. 1, 2017.

[9] M. Hanif and M. Ardakani, “Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes,” *IEEE Communications Letters*, vol. 21, no. 11, November 2017.

[10] M. H. Ardakani, M. Hanif, M. Ardakani, and C. Tellambura, “Fast successive-cancellation-based decoders of polar codes,” *IEEE Transactions On Communications*, vol. 67, no. 7, July 2019.

[11] C. Xiong, J. Lin, and Z. Yan, “Symbol-decision successive cancellation list decoder for polar codes,” *IEEE Trans. Signal Process.*, vol. 64, no. 3, 2016.

[12] P. Giard, G. Sarkis, C. Thibeault, and W. Gross, “237 gbit/s unrolled hardware polar decoder,” *Electronics Letters*, vol. 51, no. 10, 2015.

[13] K. Niu and K. Chen, "Stack decoding of polar codes," *Electronics Letters*, vol. 48, no. 12, pp. 695–697, June 2012.

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

[15] P. Trifonov, "A score function for sequential decoding of polar codes," in *Proceedings of IEEE ISIT*, Vail, USA, 2018.

[16] H. Zhou, X. Liang, C. Zhang, S. Zhang, and X. You, "Successive cancellation heap polar decoding," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

[17] H. Aurora, C. Condo, and W. J. Gross, "Low-complexity software stack decoding of polar codes," in *IEEE ISCAS*, 2018, pp. 1–5.

[18] W. Song, H. Zhou, K. Niu, Z. Zhang, L. Li, X. You, and C. Zhang, "Efficient successive cancellation stack decoder for polar codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2608–2619, 2019.

[19] H. Zhou, W. Song, W. J. Gross, Z. Zhang, X. You, and C. Zhang, "An efficient software stack sphere decoder for polar codes," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1257–1266, 2020.

[20] C. Husmann, P. C. Nikolaou, and K. Nikitopoulos, "Reduced latency ml polar decoding via multiple sphere-decoding tree searches," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, February 2018.

[21] H. Zhou, S. Tan, W. Gross, Z. Zhang, X. You, and C. Zhang, "An improved software list sphere polar decoder with synchronous determination," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, June 2019.

[22] J. Guo and A. G. i Fabregas, "Efficient sphere decoding of polar codes," in *Proceedings of IEEE ISIT*, 2015.

[23] K. Niu, K. Chen, and J. Lin, "Low-complexity sphere decoding of polar codes based on optimum path metric," *IEEE Commun. Lett.*, vol. 18, no. 2, pp. 332–335, February 2014.

[24] T.-H. Chen, K.-C. Chen, M.-C. Lin, and C.-F. Chang, "On a* algorithms for decoding short linear block codes," *IEEE Transactions On Communications*, vol. 63, no. 10, October 2015.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[26] N. Yakuba and P. Trifonov, "Multilevel buckets for sequential decoding of polar codes," in *Proceedings of IEEE PIMRC*, 2015.

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

[28] J. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 196–198, January 1972.

[29] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Trans. Signal Process.*, vol. 63, no. 19, pp. 5165–5179, October 2015.

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

[31] S. A. Hashemi, C. Condo, and M. Mondelli, "Rate-flexible fast polar decoders," *IEEE Trans. Signal Process.*, vol. 67, no. 22, 2019.

[32] A. Valembois and M. Fossorier, "Box and match techniques applied to soft-decision decoding," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 796–810, May 2004.

[33] H. T. Moorthy, S. Lin, and T. Kasami, "Soft-decision decoding of binary linear block codes based on an iterative search algorithm," *IEEE Trans. Inf. Theory*, vol. 43, no. 3, pp. 1030–1040, May 1997.

[34] G. Trofimiuk and P. Trifonov, "Block sequential decoding of polar codes," in *Proceedings of ISWCS*, Belgium, 2015, pp. 326–330.

[35] R. R. Green, "A serial orthogonal decoder," *JPL Space Program Summary*, vol. 4, no. 31-39, pp. 241–253, 1966.

[36] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 164–172, January 1972.

[37] M. P. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1379–1396, September 1995.

[38] D. E. Knuth, *The Art of Computer Programming*. Addison-Wesley, 1973, vol. 3.

[39] K. Ivanov and P. Trifonov, "Hybrid decoding of interlinked generalized concatenated codes," in *Proceedings of 9th ISTC*. Brest, France: IEEE, 2016, pp. 41–45.

[40] P. Trifonov, "Randomized chained polar subcodes," in *Proceedings of IEEE Wireless Communications and Networking Conference Workshops*. Barcelona, Spain: IEEE, 2018, pp. 292–297.

[41] L. Chandesris, V. Savin, and D. Declercq, "Dynamic-scflip decoding of polar codes," *IEEE Trans. Commun*, vol. 66, no. 6, June 2018.

[42] B. L. Gal, C. Leroux, and C. Jego, "Multi-gb/s software decoding of polar codes," *IEEE Trans. Signal Process.*, vol. 63, no. 2, January 2015.

[43] A. Balatsoukas-Stimming, A. J. Raymond, W. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Transactions On Circuits And Systems–II: Express Briefs*, vol. 61, no. 8, August 2014.

Grigorii Trofimiuk (S'15) was born in Boksitogorsk, Russia in 1994. He received the B.Sc. and M.Sc. degrees from St.Petersburg Polytechnic University in 2016 and 2018, respectively, all in computer science. He is currently working toward the Ph.D. degree at the ITMO University in St.Petersburg, Russia. His research interests include coding theory and its applications in telecommunications.

Nikolai Iakuba (S'15) was born in St.Petersburg and obtained his M.S. degree in St.Petersburg Polytechnic University. He is currently working toward the Ph.D. degree at the ITMO University in Saint Petersburg, Russia. His research interests include coding theory, especially polar and Reed-Muller codes

Stanislav Rets received the B.S. and M.S. degrees from St. Petersburg Polytechnic University, Russia, in 2015 and 2017, respectively. His research interests include coded modulation techniques based on polar codes.

Kirill Ivanov (S'16) obtained his B.S. and M.S. degrees from St. Petersburg Polytechnic University, Russia, in 2015 and 2017, respectively. Currently he is a PhD student at École polytechnique fédérale de Lausanne, Switzerland under the supervision of Prof. Rüdiger Urbanke.

His research interests include wireless communications systems and coding theory, with focus on polar and Reed-Muller codes.

Peter Trifonov (S'02,M'05) was born in St.Petersburg, USSR in 1980. He received the MSc and PhD (Candidate of Science) degrees from Saint Petersburg Polytechnic University in 2003 and 2005, and Dr.Sc degree from the Institute for Information Transmission Problems in 2018. His research interests include coding theory and its applications in telecommunications and storage systems. Currently he is a professor at the ITMO University in Saint Petersburg, Russia. He is an editor at IEEE Transaction on Communications.