Sequential Decoding of Polar Codes

V. Miloslavskaya, *Student Member, IEEE*, and P. Trifonov, *Member, IEEE*

***Abstract*—The problem of efficient decoding of polar codes is considered. A low-complexity sequential soft decision decoding algorithm is proposed. It is based on the successive cancellation approach, and it employs most likely codeword probability estimates for selection of a path within the code tree to be extended.**

***Index Terms*—Polar codes, sequential decoding, successive cancellation.**

I. Introduction

POLAR codes were recently shown to be able to achieve the capacity of a wide class of communication channels [1]. However, their performance at moderate length appears to be quite poor under successive cancellation (SC) decoding. This problem was addressed in [2], where a list decoding algorithm for polar codes was introduced. It was shown in [3], [4] that the same performance can be achieved with much lower complexity by employing a stack-based decoding algorithm.

In this paper a novel low-complexity stack-based decoding algorithm for polar codes is introduced. It employs information about the quality of not-yet-processed frozen bit subchannels to reduce the number of times the decoder switches between different paths in the code tree, while processing a received vector. The proposed approach avoids probability-domain calculations, required by the algorithms given in [2]–[4], which are very difficult to implement in hardware.

II. Background

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. By a_i^j we will denote the sequence $(a_i, a_{i+1}, \ldots, a_j)$. Any codeword of a polar code can be represented as $c_0^{n-1} = u_0^{n-1} G_n$, where u_0^{n-1} is 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 [4] to set $u_i,\ i \in \mathcal{F}$, to some linear function of u_0^{i-1} (dynamic frozen symbols). This enables one to obtain codes with higher minimum distance, while still allowing one to employ the SC decoding algorithm and its variations. For the sake of simplicity, the proposed decoding algorithm will be derived for the case of classical polar codes. Its extension to the case of codes with dynamic frozen symbols is straightforward.

The decoding problem consists in identifying u_0^{n-1} which maximizes $P_{U_0^{n-1}|Y_0^{n-1}}(u_0^{n-1}|y_0^{n-1})$, such that $u_i = 0$ for all $i \in \mathcal{F}$, where U_j is a random variable corresponding to the j-th input symbol of the polarizing transformation, and Y_j is a random variable corresponding to the j-th received symbol. In the i-th phase of the classic SC decoding algorithm probabilities $P_{U_0^i|Y_0^{n-1}}(u_0^i|y_0^{n-1}) = (P_{Y_0^{n-1},U_0^{i-1}|U_i}(y_0^{n-1}, u_0^{i-1}|u_i))/(2P_{Y_0^{n-1}}(y_0^{n-1}))$, $u_i \in \{0,1\}$, are calculated. The decoder makes decisions $\hat{u}_i = \arg\max_{u_i \in \{0,1\}} P_{Y_0^{n-1},U_0^i|U_i}(y_0^{n-1}, u_0^i|u_i)$ for $i \notin \mathcal{F}$ and $\hat{u}_i = 0$ otherwise. $\hat{u}_0, \ldots, \hat{u}_i$ are used instead of the true values of $u_0, \ldots, u_i$ while computing the probabilities at subsequent steps. It was shown in [1] that the SC algorithm can be implemented with complexity $O(n \log_2 n)$.

A major drawback of the SC algorithm is that it cannot correct errors which may occur at early phases of the decoding process. This problem is solved in stack/list algorithms by keeping a list of the most probable paths of different lengths. The stack decoding algorithm [3], [4] explores paths within the code tree. A path of length i is identified by values u_0^{i-1}. Each path has an associated score. All paths are stored in a stack (priority queue). At each iteration the decoder selects for extension path u_0^{i-1} with the largest score, and performs the i-th phase of SC decoding. That is, if $i \in \mathcal{F}$, the path is extended to obtain $(u_0^{i-1}, 0)$, and the extended path is stored in the stack together with its score. Otherwise, the path is cloned to obtain new paths $(u_0^{i-1}, 0)$ and $(u_0^{i-1}, 1)$, which are stored in the stack together with their scores. In order to keep the size of the stack limited, 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. Decoding terminates as soon a path of length n appears at the top of the stack, or the stack becomes empty. Hence, the worst case complexity of stack decoding is given by $O(Ln \log_2 n)$. Average decoding complexity depends on how path scores are defined.

III. Proposed Algorithm

A. Estimating Path Probability

The objective of the decoder is to find path u_0^{n-1} maximizing $P_{U_0^{n-1}|Y_0^{n-1}}(u_0^{n-1}|y_0^{n-1})$. Let us estimate the probability $T(u_0^i, y_0^{n-1}) = \max_{u_{i+1}^{n-1} : u_{i+1,\mathcal{F}}^{n-1} = \mathbf{0}} P_{U_0^{n-1}|Y_0^{n-1}}(u_0^{n-1}|y_0^{n-1})$ of transmission of the most probable codeword of the polar code in the code subtree given by prefix u_0^i, assuming that u_0^i values are correct. Here $a_{j,\mathcal{D}}^h$ is a subvector of vector a_j^h consisting of elements $a_s,\ s \in \mathcal{D} \cap \{j, \ldots, h\}$. This estimate will be used to predict which path u_0^i may correspond to the solution of the decoding problem.

It is difficult to compute $T(u_0^i, y_0^{n-1})$ at phase i of the SC decoder. Let $v[j]_0^{n-1},\ 0 \le j < 2^{n-i-1}$, be different paths in the

Manuscript received December 19, 2013; accepted April 24, 2014. Date of publication May 13, 2014; date of current version July 8, 2014. This work was supported by the Russian Foundation for Basic Research under Grant 12-01-00365-a. The associate editor coordinating the review of this paper and approving it for publication was M. F. Flanagan.

The authors are with the Department of Distributed Computing and Networking, Saint-Petersburg State Polytechnical University, Saint-Petersburg 194021, Russia (e-mail: veram@dcn.icc.spbstu.ru; petert@dcn.icc.spbstu.ru).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LCOMM.2014.2323237

subtree, such that $v[j]_0^i = u_0^i$ and $v[j]_{i+1}^{n-1} \in \{0,1\}^{n-i-1}$. Let J be a random variable, which is equal to j if the most probable codeword of the polar code corresponds to path $v[j]_0^{n-1}$. Observe that $J = j$ implies that $v[j]_h = 0,\ h \in \mathcal{F}$. We propose to estimate $T(u_0^i, y_0^{n-1})$ as $E_J[P_{U_0^{n-1}|Y_0^{n-1}}(v[J]_0^{n-1}|y_0^{n-1})]$. Note that increasing i reduces the number of possible values of J, improving thus the accuracy of such estimate.

Let $\alpha = \arg\max_{0\le j<2^{n-i-1}} P_{U_0^{n-1}|Y_0^{n-1}}(v[j]_0^{n-1}|y_0^{n-1})$. Event $J = \alpha$ is equivalent to $v[\alpha]_h = 0, h \in \mathcal{F}$. Hence,

$$\begin{aligned} T\left(u_0^i, y_0^{n-1}\right) &\approx E_J\left[P_{U_0^{n-1}|Y_0^{n-1}}\left(v[J]_0^{n-1}|y_0^{n-1}\right)\right] \\ &= \sum_{j=0}^{2^{n-i-1}-1} P_{U_0^{n-1}|Y_0^{n-1}}\left(v[j]_0^{n-1}|y_0^{n-1}\right) P\{J=j\} \\ &\ge \underbrace{P_{U_0^{n-1}|Y_0^{n-1}}\left(v[\alpha]_0^{n-1}|y_0^{n-1}\right)}_{R(u_0^i, y_0^{n-1})} P\{J=\alpha\}. \end{aligned} \tag{1}$$

Average value of $P\{J = \alpha\}$ over all possible received sequences is equal to the probability of the most probable path u_0^{n-1} of prefix u_0^i having zeroes in positions $j \in \mathcal{F}$. It is lower bounded by the probability $\hat{\Omega}(i)$ that the SC decoder, which starts from u_0^i and does not take into account any freezing constraints, makes decisions $u_j = 0,\ j > i,\ j \in \mathcal{F}$, i.e., makes no errors in these positions.

Hence, one obtains the following estimate for $T(u_0^i, y_0^{n-1})$:

$$\hat{T}\left(u_0^i, y_0^{n-1}\right) = R\left(u_0^i, y_0^{n-1}\right)\hat{\Omega}(i) \tag{2}$$

where $\hat{\Omega}(i) = \prod_{j\in\mathcal{F}, j>i}(1 - P_j)$, and P_j is the j-th subchannel error probability, provided that exact values of all previous bits $u_{j'}$, $j' < j$, are available. For any given channel P_j can be pre-computed offline using density evolution [5], [6]. $\hat{\Omega}(i)$ depends only on n, $\mathcal{F}$ (i.e., the code being considered), channel properties and phase i.

At each iteration of the stack decoding algorithm one can select for extension the path with the largest $\hat{T}(u_0^i, y_0^{n-1})$. Observe that $\hat{\Omega}(i)$ increases with i, while $R(u_0^i, y_0^{n-1})$ decreases with i for any given u_0^i. Hence, given two paths with the same value of $R(u_0^i, y_0^{n-1})$, the decoder would prefer the longer one. This approach enables one to compare paths u_0^i with different lengths, and prevent the decoder from switching frequently between different paths.

Let us show how probabilities $R(u_0^i, y_0^{n-1})$ can be computed. According to [1], the encoding operation $c_0^{n-1} = u_0^{n-1}G_n$ can be represented via recursive expressions $c_0^{n/2-1} = (u_{0,even}^{n-1} \oplus u_{0,odd}^{n-1})G_{n/2}$ and $c_{n/2}^{n-1} = u_{0,odd}^{n-1}G_{n/2}$, where $u_{0,even}^i$ and $u_{0,odd}^i$ are subsequences of u_0^i consisting of elements with even and odd indices, respectively. Thus, encoding of an input sequence of length n reduces to encoding of two sequences of length $n/2$. This implies that

$$\begin{aligned} R\left(u_0^{2i}, y_0^{n-1}\right) &= \max_{u_{2i+1}\in\{0,1\}} R\left(u_{0,even}^{2i+1} \oplus u_{0,odd}^{2i+1}, y_0^{n/2-1}\right) \\ &\quad \cdot R\left(u_{0,odd}^{2i+1}, y_{n/2}^{n-1}\right), \end{aligned} \tag{3}$$

$$\begin{aligned} R\left(u_0^{2i+1}, y_0^{n-1}\right) &= R\left(u_{0,even}^{2i+1} \oplus u_{0,odd}^{2i+1}, y_0^{n/2-1}\right) \\ &\quad \cdot R\left(u_{0,odd}^{2i+1}, y_{n/2}^{n-1}\right). \end{aligned} \tag{4}$$

The initial value for these recursive expressions is given by $R(b, y_j) = P_{X_j|Y_j}(b|y_j),\ b \in \{0,1\}$, where X_j is a random variable corresponding to the j-th codeword symbol.

B. Efficient Implementation

To obtain efficient implementation of the proposed method we follow [2]. Namely, we present memory-efficient techniques for computing $R(u_0^i, y_0^{n-1})$. Furthermore, we derive log-domain implementation of the algorithm, which employs only summation and comparison operations without any performance loss.

Let λ, ϕ, and β denote layer, phase and branch number, respectively, where $0 \le \beta < 2^{m-\lambda}$. At each decoding iteration the only output needed is $R((u_0^i, b), y_0^{n-1}),\ b \in \{0,1\}$, so it is associated with branch number $\beta = 0$. It can be computed recursively as follows. For $\lambda > 0$ one should compute $R(\hat{u}_0^{\phi-1}, y_0^{\Lambda-1})$, where $\Lambda = 2^\lambda$. Denote $\psi = \lfloor \phi/2 \rfloor$. Let $R(\hat{u}_{0,even}^{2\psi-1} \oplus \hat{u}_{0,odd}^{2\psi-1}, y_0^{\Lambda/2-1})$ and $R(\hat{u}_{0,odd}^{2\psi-1}, y_{\Lambda/2}^{\Lambda-1})$ be associated with branch number 2β and $2\beta+1$, respectively.

Let $\langle \phi, \beta \rangle_\lambda = \phi + 2^\lambda \beta$

$$R_\lambda\left[\langle \phi, \beta \rangle_\lambda\right][b] = R\left(\left(\hat{u}_0^{\phi-1}, b\right), y_0^{\Lambda-1}\right).$$

Similarly to [2], it will be possible to drop index ϕ. Therefore, for brevity this quantity will be denoted by $R_\lambda[\beta][b]$.

With array notation expressions (3) and (4) become

$$R_\lambda[\beta][b] = \begin{cases} \max\limits_{d\in\{0,1\}} R_{\lambda-1}[2\beta][b\oplus d] R_{\lambda-1}[2\beta+1][d], & \phi \text{ even}, \\ R_{\lambda-1}[2\beta]\left[C_\lambda[\beta][0]\oplus b\right] R_{\lambda-1}[2\beta+1][b], & \phi \text{ odd}, \end{cases}$$

where $C_\lambda[\beta][b]$ is defined in the same way as in [2].

Let us make a change of variable $S_\lambda[\beta] = \ln(R_\lambda[\beta][0]/R_\lambda[\beta][1])$. Then, one obtains

$$\begin{aligned} S_\lambda[\beta] &= \begin{cases} Q\left(S_{\lambda-1}[2\beta], S_{\lambda-1}[2\beta+1]\right), & \phi \text{ even}, \\ (-1)^{C_\lambda[\beta][0]} S_{\lambda-1}[2\beta] + S_{\lambda-1}[2\beta+1], & \phi \text{ odd}. \end{cases} \\ R_\lambda[\beta][1] &= R_{\lambda-1}[2\beta][1] R_{\lambda-1}[2\beta+1][1] \exp\left(Z_\lambda[\beta]\right), \end{aligned} \tag{5}$$

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

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

It can be seen that

$$\ln R_m[0][1] = \left(\sum_{j=0}^{2^m-1} \ln P_{X_j|Y_j}(1|y_j)\right) + \left(\sum_{\lambda=1}^{m} \sum_{\beta=0}^{2^{m-\lambda}-1} Z_\lambda[\beta]\right).$$

So, $D_m = \ln R_m[0][1]$ can be computed as follows. Let $D_0 = \sum_{j=0}^{2^m-1} \ln P_{X_j|Y_j}(1|y_j)$. For $\lambda = 1, \ldots, m$ one obtains

$$D_\lambda = D_{\lambda-1} + \sum_{j=0}^{2^{m-\lambda}-1} Z_\lambda[j]. \tag{7}$$

However, since D_0 does not depend on u_0^i and $\ln R_m[0][1] = D_m$, $\ln R_m[0][0] = D_m + S_m[0]$, one can set $D_0 = 0$ in this recursion, so $\ln R_m[0][b],\ b \in \{0,1\}$, are changed by the same value for all u_0^i.

The above described calculations can be implemented as follows. Let l be the index of a path corresponding to some $u_0^{\phi_l-1}$. For each l one should keep the values $S_{l,\lambda}[\beta]$, $D_{l,\lambda}$, as

```
RecursivelyCalcS(l, λ, φ)
 1  if (λ = 0)  return
 2  if (φ even)  RecursivelyCalcS(l, λ − 1, ⌊φ/2⌋)
 3  D_{l,λ} ← D_{l,λ−1}
 4  for (β = 0, ..., 2^{m−λ} − 1)
 5    if (φ even)
 6      S_{l,λ}[β] ← Q(S_{l,λ−1}[2β], S_{l,λ−1}[2β + 1])
 7      D_{l,λ} ← D_{l,λ} + max(S_{l,λ−1}[2β], S_{l,λ−1}[2β + 1])
 8    else
 9      if (C_{l,λ}[β][0] = 0)
10        S_{l,λ}[β] ← S_{l,λ−1}[2β] + S_{l,λ−1}[2β + 1]
11      else
12        S_{l,λ}[β] ← S_{l,λ−1}[2β + 1] − S_{l,λ−1}[2β]
13        D_{l,λ} ← D_{l,λ} + S_{l,λ−1}[2β]
```

Fig. 1. Calculation of path score.

```
Decode(y_0^{n−1}, L, Θ)
 1  InitializeDataStructures()
 2  l ← AssignInitialPath()
 3  φ_l ← 0
 4  S_{l,0}[β] ← ln (P_{X_β|Y_β}(0|y_β) / P_{X_β|Y_β}(1|y_β)),   β = 0, ..., n − 1
 5  D_{l,0} ← 0
 6  push(0, l); P ← 1; q ← (0, ..., 0)
 7  while (P > 0)
 8    (M, l) ← PopMax(); P ← P − 1
 9    q_{φ_l} ← q_{φ_l} + 1
10    if (φ_l = n)  return (C_{l,0}[0, 0], ..., C_{l,0}[n − 1, 0])
11    RecursivelyCalcS(l, m, φ_l)
12    if (φ_l ∈ F)
13      C_{l,m}[0, φ_l mod 2] ← 0
14      push(S_{l,m}[0] + D_{l,m} + Ω̂_{ln}(φ_l), l); P ← P + 1
15      if (φ_l odd)  RecursivelyUpdateC(l, m, φ_l)
16      φ_l ← φ_l + 1
17    else
18      while (P ≥ Θ − 1)
19        (M, l) ← PopMin()
20        KillPath(l); P ← P − 1
21      C_{l,m}[0, φ_l mod 2] ← 0
22      l′ ← ClonePath(l)
23      C_{l′,m}[0, φ_l mod 2] ← 1
24      push(S_{l,m}[0] + D_{l,m} + Ω̂_{ln}(φ_l), l)
25      push(D_{l,m} + Ω̂_{ln}(φ_l), l′); P ← P + 2
26      if (φ_l odd)
27        RecursivelyUpdateC(l, m, φ_l)
28        RecursivelyUpdateC(l′, m, φ_{l′})
29      φ_{l′} ← φ_l ← φ_l + 1
30    if (q_{φ_l} ≥ L)
31      for each path l′ in the priority queue
32        if (φ′_l < φ_l)
33          Erase(l′); KillPath(l′); P ← P − 1
```

Fig. 2. Sequential decoding of polar codes.

well as $C_{l,\lambda}[\beta][b]$, $0 \le \lambda \le m$, $0 \le \beta < 2^{m-\lambda}$, $b \in \{0,1\}$. The calculations given by (5)–(7) are implemented in algorithm $RecursivelyCalcS$ shown in Fig. 1. Call $RecursivelyCalcS(l, m, \phi_l)$ reuses the intermediate values obtained at previous calls for the same l.

Fig. 2 presents the proposed sequential decoding algorithm. It performs the same initialization as Tal-Vardy list decoding algorithm, where probability array P is replaced with array S and additionally array D is created. The same lazy copying shared memory data structures as in [2] are used for these arrays. Elements $S_{l,0}[j]$, $j = 0, \ldots, n-1$, are initialized with LLRs $\ln(P_{X_j|Y_j}(0|y_j))/(P_{X_j|Y_j}(1|y_j))$. The decoder keeps paths in a stack (priority queue) ordered according to their scores. At each iteration a path with the largest score M is extracted from the stack. LLRs $S_{l,m}[0]$, as well as $D_{l,m}$, are computed for this path. If path phase ϕ_l corresponds to a frozen symbol, the path is extended[1] with 0. Otherwise, the path is cloned and extended with 0 and 1. Scores of extended paths include term $\hat{\Omega}_{ln}(\phi) = \ln \hat{\Omega}(\phi)$, which is given by (2). The algorithm makes use of procedure $RecursivelyUpdateC$, which is the same as the one in [2] (Algorithm 15), except that it is employed for a path with number l.

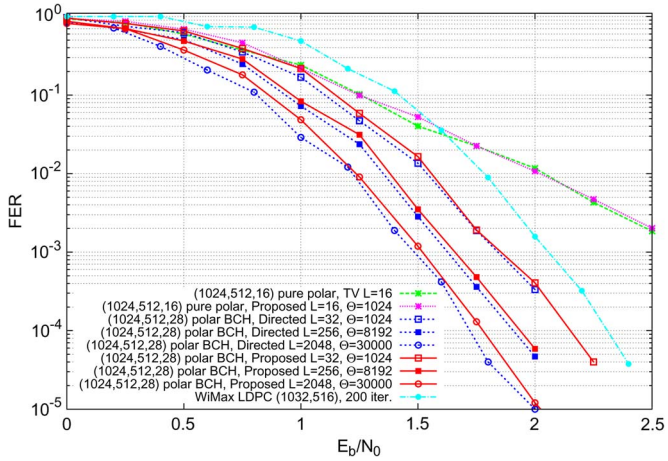

Fig. 3. Performance of (1024, 512) polar codes.

If stack capacity Θ is about to be exceeded, paths with low scores are eliminated in line 20 of Fig. 2. If the number of different paths in the code tree extended till phase q_{ϕ_l} exceeds L, then short paths are removed in line 33 of Fig. 2.

Observe that the proposed algorithm makes use of only addition, subtraction and comparison operations. Furthermore, array $P_{l,\lambda}[\beta][b]$, used in Tal-Vardy algorithm, is replaced with the array $S_{l,\lambda}[\beta]$ which has half of the size. Each path considered by the algorithm requires at most n memory cells to store real values $S_{l,\lambda}[\beta]$, $m+1$ real values $D_{l,\lambda}$, and $2n$ bits $C_{l,\lambda}[\beta][b]$. Therefore, the total space requirements are $O(\min(Ln, \Theta)(3n + m + 1))$, which is substantially more than that of, e.g., existing LDPC decoders. Reduction of memory consumption is subject of further research.

C. Further Improvements

The decoding complexity can be further reduced as follows:

- Call $RecursivelyCalcS$ only for $\phi_l \notin \mathcal{F}$. If a path is extended with a frozen symbol, its old score value may be preserved. Observe that this requires some modifications in $RecursivelyCalcS$ in order to ensure that all its source data are available.
- As soon as all frozen symbols are processed, one can switch to hard decision decoding.

[1] If the algorithm is applied to a polar code with dynamic frozen symbols, one should extend the path with the value given by the dynamic freezing constraint, and adjust appropriately line 13 of Fig. 2.

TABLE I
AVERAGE COMPLEXITY OF DECODING ALGORITHMS, $\times 10^3$ REAL OPERATIONS

E_b/N_0, dB	Proposed approach						Directed search		Belief propagation	
	Sum			Comparison			Sum	$\ln(1+e^{-x})$	Sum	$\ln\tanh(\frac{x}{2})$
	$L=32$	$L=256$	$L=2048$	$L=32$	$L=256$	$L=2048$	$L=32$	$L=32$	≤ 200 iter.	≤ 200 iter.
0.5	133	752	4265	218	1224	6968	475	88	2333	1112
1	73	286	1232	122	477	2065	367	68	1469	722
1.5	32	88	267	54	151	461	218	40	394	185
2	18	27	42	31	48	74	119	22	140	62

IV. NUMERICAL RESULTS

Fig. 3 presents simulation results illustrating the performance of the proposed decoding algorithm, as well as that of Tal-Vardy list decoding algorithm [2], and the directed search algorithm [4]. The results are reported both for the case of an (1024,512,16) pure polar code and an (1024,512,28) polar code with dynamic frozen symbols obtained as a subcode of an (1024,893,28) extended BCH code, as well as an (1032,516) LDPC code. It can be seen that the perfomance of the proposed algorithm for the case of pure polar code is identical to that of Tal-Vardy list decoding algorithm. It must be recognized that the performance of this code is limited by its poor minimum distance, and increasing list size L does not provide any gain. Much better performance can be obtained with an (1024,512,28) code. It can be seen that the proposed decoding algorithm provides slightly worse performance than the directed search algorithm with the same L.

Table I illustrates the complexity of the proposed approach, log-domain implementation of directed search algorithm for polar codes, as well as belief propagation algorithm for LDPC codes with at most 200 iterations. It can be seen that the proposed algorithm requires much smaller number of arithmetic operations with real numbers. Furthermore, it avoids evaluation of non-linear functions, which are used in log-domain implementation of belief propagation algorithm, and some other decoding algorithms for polar codes [2]–[4].

Fig. 4 presents correct path scores $\ln \hat{T}(u_0^i, y_0^{n-1})$ (see expression (2)) for a few instances of the decoding problem. It can be seen that one obtains score values close to the final ones at very early decoding phases, after the decoder processes the initial blocks of frozen symbols (phases corresponding to frozen symbols are designated by $+$). This prevents the decoder from switching to incorrect paths.

V. CONCLUSION

In this paper a novel decoding algorithm for polar codes was proposed. Its complexity was shown to be substantially lower compared to the existing list and stack decoding algorithms. Complexity reduction is achieved at the cost of negligible performance degradation. However, it enables one to use much larger list size, allowing thus to obtain significant performance gain at the same complexity level as state-of-the-art algorithms.

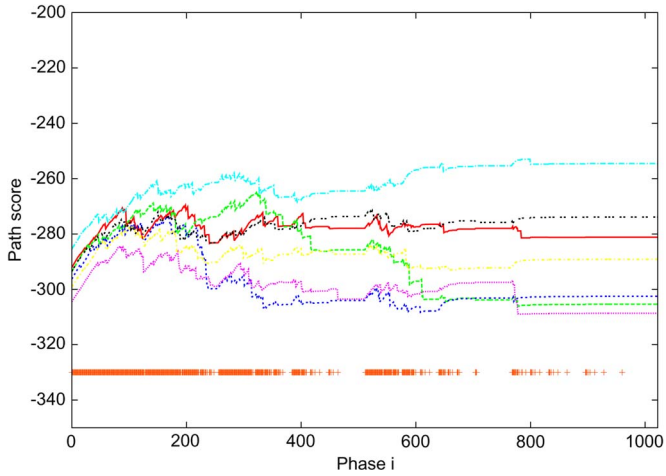

Fig. 4. Correct path scores for (1024, 512, 28) polar code.

Min-sum list decoding algorithm, similar to the proposed one, was considered in [7]. However, it operates only with fixed-length paths, and has therefore higher average computational complexity than the proposed method, but smaller memory consumption.

REFERENCES

[1] E. Arikan, "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, Jul. 2009.
[2] I. Tal and A. Vardy, "List decoding of polar codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 1–5.
[3] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, Oct. 2012.
[4] P. Trifonov and V. Miloslavskaya, "Polar codes with dynamic frozen symbols and their decoding by directed search," in *Proc. IEEE Int. Workshop Inf. Theory*, Sep. 2013, pp. 1–5.
[5] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.
[6] R. Pedarsani, S. H. Hassani, I. Tal, and E. Telatar, "On the construction of polar codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 11–15.
[7] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," in *Proc. 39th Int. Conf. Acoust., Speech, Signal Process.*, 2014, pp. 1–5.