IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 5 May 2024, accepted 17 June 2024, date of publication 19 June 2024, date of current version 27 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3416826

RESEARCH ARTICLE

Fast Polar Decoding With Successive Cancellation List Creeper Algorithm

ILYA TIMOKHIN, (Member, IEEE), AND FEDOR IVANOV, (Member, IEEE)
Department of Cyber-Physical Systems Information Security, National Research University Higher School of Economics, 101000 Moscow, Russia

Corresponding author: Ilya Timokhin (is.timokhin@hse.ru)

This work was supported by the Basic Research Program, National Research University Higher School of Economics (HSE), in 2024.

ABSTRACT Polar codes have emerged as a focal point in the field of error-correcting codes, owing to their remarkable capacity-achieving characteristics and their relevance in various modern communication systems. The basic successive cancellation (SC) approach is not optimal to use in terms of the trade-off between performance and decoding complexity. SC-Creeper algorithm performs better with about the same low complexity as the SC version of the algorithm. However, the SC-Creeper algorithm did not have the ability to use the candidate list as a measure to improve performance and refine the search for the true codeword. To compare with successive cancellation list (SCL) approach and the ability to use more computing memory, the SCL-Creeper method was developed, using two additional lists. This method can also be used as a development of Fano algorithms for polar codes (mainly, Fano decoding in polar decoding does not use lists). This paper addresses the challenge of computational complexity in polar code decoding by integrating a list structure with the SC-Creeper algorithm. Building on prior research that introduced the concept of SC-Creeper, the study focuses on enhancing error correction performance while mitigating computational burden. The first chapters describe the polar encoding process and basic decoding technologies, then discuss the basic Creeper algorithm. In the following chapters, the authors describe a modified version of the two-list Creeper approach (that is, the SCL-Creeper version of the algorithm). Extensive simulations and numerical analysis presented in the paper underscore the tangible advantages of this novel decoding strategy. Leveraging the basic list algorithm, renowned for its superior error correction capabilities, the research explores the integration of Creeper to systematically prune unnecessary decoding paths. The resulting SCL-Creeper hybrid approach aims to strike a balance between error correction efficiency and computational complexity. Finally, the optimal selection of parameters for the SCL-Creeper approach and future directions in the research of the list version of the fast Creeper algorithm are discussed.

INDEX TERMS Polar codes, decoding, creeper, reliability, fast decoding.

I. INTRODUCTION

Polar codes [1] have established themselves as a transformative class of error-correcting codes, with theoretical promises that approach the Shannon capacity for symmetric binary discrete memoryless channels as the code length increases. The practical implementation of polar codes for short to moderate code lengths has been constrained by the computational complexity associated with the conventional successive cancellation (SC) decoder. In response to these challenges, researchers have been motivated to explore innovative decoding paradigms, such as the successive cancellation list (SCL) decoding approach [7], which retains a list of potential candidate paths for further examination. This SCL technique has yielded improvements in error correction performance at the cost of some additional computational overhead.

In addition to the SCL decoder, there is a similar approach that uses a stack to store paths, the management of which allows for more efficient use of memory and the use of fewer candidates than in the case of SCL. This approach is called SCS (successive cancellation stack [2]) and provides efficient decoding with space improvements.

Building upon the success of SCL and SCS decoding approaches, this paper introduces a novel advancement in polar code decoding – successive cancellation list Creeper

The associate editor coordinating the review of this manuscript and approving it for publication was Cesar Vargas-Rosales.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

(SCL-Creeper) algorithm. The SCL-Creeper algorithm integrates the strengths of SCL decoding with a path-wise traversal strategy inspired by the Creeper algorithm originally developed for convolutional codes [5]. For reliability and efficient tree traversal, this algorithm uses two stacks that work on the principle of SCS decoding, and a list is used to store path metrics and generate possible candidates, as in the SCL approach. Thus, SCL-Creeper, inspired by two classical polar decoding approaches, is a hybrid of a robust and fast decoder.

Also for comparison, we consider an algorithm that uses flipping decoding with additional attempts and a list of possible candidates, as in the basic SCL case. Successful cancellation flip (SCF) and list-based (SCLF) techniques are studied for comparison with the list version of the Creeper approach presented in this work and are also analyzed in terms of performance and complexity parameters.

In this paper, we delve into a comprehensive exploration of the SCL-Creeper algorithm. We provide a detailed analysis of the algorithm's performance, comparing it to basic polar decoding techniques, and highlight its potential to revolutionize polar code decoding for different code lengths. Additionally, we present the results of a computer simulation that demonstrate the SCL-Creeper algorithm's effectiveness in improving error correction capabilities and its suitability for real-world applications. SCL-Creeper algorithm is compelling solution for harnessing the full potential of polar codes in a wide range of communication and data storage systems.

II. RELATED WORKS

The topic of optimal polar decoding in terms of computational complexity and performance has been extensively explored in numerous studies [29]. For instance, a study [23] on the SC-Fano algorithm allows us to draw conclusions regarding various decision-making processes involved in traversing a decoding tree. Furthermore, it introduces the intriguing concept of acceleration resulting from specific nodes.

The SC-Fano flipping decoder [24] not only reduces computational complexity within a single decoding attempt but also enhances performance through an adaptive algorithm. This approach exemplifies the symbiosis of two strategies: the allocation of additional resources (such as decoder attempts, CRCs, candidate lists, etc.) and the utilization of Fano decoding principles alongside decision-making metrics.

The isolation of special nodes suitable for accelerating decoding for any codewords is an intriguing and extensive topic addressed in the literatures [25] and [27]. Alongside conventional decoding methods, accelerated algorithmic variants are also explored. However, for list methods, these special nodes require modification, as they pertain to metrics for calculating the probability of a candidate's inclusion in the list. Some optimizations [26] concerning irregular polar codes enable achieving relatively low computational complexity using additional nodes without a list, leveraging the structure of polar code construction. This facilitates the acceleration of decoding for a specific class of polar codes, distinct from those considered in the present study.

Furthermore, researchers propose expediting the convergence of belief propagation decoding methods [28]. In this scenario, reducing the number of iterations (i.e., achieving fast convergence) results in a decrease in computational complexity asymptotically. Hence, employing the belief propagation strategy in certain cases serves as a technique to reduce time complexity. However, a primary challenge associated with this approach is the potential compromise in performance due to fewer decoding iterations.

III. POLAR ENCODING AND DECODING CONCEPTS

Let's assume polar code with a finite code length $N = 2^n$ ($n \in \mathbf{Z}^+$), information length of K and reliability bits sequence $I \in \{0, \ldots, N-1\}$, $|I| = K$. Subchannels with intermediate reliability fall between silent ones, which have high reliability, and noisy ones, which have low reliability. Subchannels F not included in I will be called frozen. Its values will be zeros. By notation $\mathbf{x}_i^j$, $i \leq j$ authors will mean the vector $(x_i, x_{i+1} \ldots, x_j)$. Encoding procedure can be described by the next equation:

$$\mathbf{x}_0^{N-1} = \mathbf{d}_0^{N-1}\mathbf{G}_N, \tag{1}$$

where $\mathbf{d}_0^{N-1}$ is initial message vector with binary elements, $\mathbf{x}_0^{N-1}$ is encoded vector, $\mathbf{G}_N = \mathbf{F}^{\otimes n}$, the matrix $\mathbf{F} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ specifies the standard Arican transform, and $\otimes n$ is the n-th Kronecker power of the matrix $\mathbf{F}$.

The authors will denote an input LLRs vector (log-likelihood ratio) as $y_0^{N-1} = (y_0, y_1, \ldots, y_{N-1})$. The result of decoding (or the only one node of decoding) in the meaning of likelihood data calls "output LLRs vector" and denotes as $\mathbf{a}_0^{N-1}$. Output decoded message based on the output LLRs vector will be denoted as $\overline{\mathbf{d}}_0^{N-1}$.

SC-decoding algorithm can be thought of as a greedy tree search algorithm for polar code. Of the two possible values of the information bit, only one with the highest probability is selected for subsequent processing. SCL method adopts a distinct strategy compared to the conventional SC approach. Instead of directly estimating the message vector, SCL generates a list of potential L candidates and identifies the correct one in the final stage. Specialized path metric (PM) metric quantifies the likelihood of each path. Computational complexity of such algorithm with fixed list size L is estimated [7] as $\mathcal{O}(LN\log N)$ and space complexity – $\mathcal{O}(LN)$.

To discern the path closest to the source word in terms of probability, additional CRC (cyclic redundancy check) bits are incorporated. Through an evaluation of CRC values, the correct path among the candidates is determined, a technique

known as CRC-aided SCL (CA-SCL), which significantly enhances performance and path selection accuracy.

Stack decoding algorithm represents an optimization of the SCL decoder, primarily aimed at reducing computational complexity. SCS introduces two key parameters: L, which is akin to the list size in SCL and restricts the search width during decoding, and D, representing the stack size. The complexity of computation with stack is equals to $\mathcal{O}(LN\log N)$ and its space complexity is $\mathcal{O}(DN)$.

Flipping-based method described in [21] is a generalization of the SC-decoder with CRC validation to improve its error correction performance. This method does not guarantee finding the correct codeword and is parameterized by T attempts of SC-decoder refining. This algorithm has a significant improvement in performance compared to the SC-decoder, but its complexity $\mathcal{O}(WN\log N)$ — it invokes the SC algorithm W times.

Due to the incompatibility between SC metrics and path metrics from the SCL decoder, it is challenging to determine how to seamlessly integrate the SCL algorithm into the SCF strategy. This SCLF algorithm [20] also uses the parameter T, as in the SCF approach, which represents the maximum number of bit flip attempts. Instead of iterative decoding with a one-level bit-flipping procedure using the SC method, it employs the SCL function with CRC bits to obtain the $\widehat{y}_1^N$ vector.

This method achieves slightly improved performance for the SCF approach [19], and also illustrates an increase in the reliability metric compared to the basic SCL approach. Next, we will consider flexible and generalized approaches to the SCLF concept to achieve the optimal decoding performance.

IV. CREEPER ALGORITHM

Novel decoding strategy known as SC-Creeper is founded [3] upon the convolutional Creeper approach. SC-Creeper melds aspects of stack-based decoding with the Fano decoding [4], utilizing the Fano metric as the path metric (PM). The key distinction between SC-Creeper and the Fano approach lies in the computation of the dynamic threshold denoted as T.

In SC-Creeper, a last-in-first-out (LIFO) stack, denoted as $\mathcal{N}$, takes center stage as the primary data structure for assessing likelihood paths. SC-Creeper introduces essential definitions related to the core tree structure. For a given node $v_i^{(t)}$, representing the current state at level t, SC-Creeper avoids the need to maintain the entire code tree in memory. The stack $\mathcal{N}$ in SC-Creeper has two main operations: header's element v from this stack, denoted as $V : \mathcal{N} \to v$ and obtaining the position of element p_v from node v in stack, this function denotes as $Pos : \mathcal{N} \to p_v$.

When the path metric surpasses the threshold T, it is identified as a "valid node". All the valid nodes are maintained in the $\mathcal{T}$ stack. One of the most powerful features [8] of the SC-Creeper approach is its computational complexity, comparable to the SC-algorithm with complexity optimizations, and uses $\mathcal{O}(N)$ operations. Space complexity is also linear: $\mathcal{O}(N)$.

V. OUR CONTRIBUTION: SCL-CREEPER

In this section authors present SCL-Creeper decoding algorithm. The basic SC-Creeper operates on two stacks with the ability to refine the metric values for each path. Let us assume that the decoding performance can be improved as it was for convolutional list decoding [9]. The main idea is to use not only the SCS-like approach, but also SCL.

To calculate the path metric for stacks, this work uses the Fano metric called PMF (Fano-based path metric), which goes back to the work on Fano list decoding. As in the case of the Creeper method, PMF_i metric is calculated iteratively:

$$\mathrm{PMF}_i = \begin{cases} \mathrm{PMF}_{i-1} + \log \dfrac{\mathrm{P}(d_i | \mathbf{y}_0^{N-1}, \overline{\mathbf{d}}_0^{N-1})}{1 - p_i}, & i > -1 \\ 0, & i = -1 \end{cases} \tag{2}$$

Here we consider a probabilistic scheme, where for each i-th channel there is an error probability p_i. Therefore, for the initial version of the SCL-Creeper algorithm, it was decided to use the basic path metric described in Equation 2. $\mathrm{PM_i}$ depends on a_i value from decoding LLRs output. It is possible to use both PM and PMF metric for SCL-Creeper algorithm; generally speaking, PMF metric also comes from log-likelihood ratios values:

$$a_i = \log \frac{P\left(d_i = 0 \,\middle|\, \mathbf{y}_0^{N-1}, \overline{\mathbf{d}}_0^{N-1}\right)}{P\left(d_i = 1 \,\middle|\, \mathbf{y}_0^{N-1}, \overline{\mathbf{d}}_0^{N-1}\right)} \tag{3}$$

The next question for implementing the algorithm is how to choose the dynamic threshold depending on the length of the list? To implement the algorithm, the following value was chosen for the threshold T: $T = \lfloor \log_2(N/L) \rfloor$.

This choice was due to the fact that for a list decoder with CRC checks, going through the entire tree (which corresponds to the value $T \approx 0$) is not an optimal solution, because such a pass is calculated to be equivalent to one iteration of SC-Creeper without a list. In the case of additional candidates and cyclic checks, you can select a threshold value based on the number of levels in the code tree.

This value can also be interpreted as follows: for example, with list size $L = 8, N = 64$, the threshold value is $T = 3$, that is, out of six levels of the code tree, the PMF value will be refined for three (where the metrics from SC-Creepers do not reach the threshold T), and for three more levels the refinement will occur not according to the Fano metric, but according to the path metric from SCL.

The algorithm consists of filling the stack/list and clearing the stack/list, both of which work in parallel. Thus, as with the basic SCL approach, this method stores child nodes as potential codeword candidates. However, unlike the basic implementation of the list method, SCL-Creeper uses a stack to check whether the current codeword should be

included in the list. If both children for some node are valid and PMF of some child is greater than any other metric calculated before, let's this node be a T-node. The current filling of the stack depends on the values of the metrics, and the filling of the list depends on the current state of the stack. Similarly, depending on whether the current pointer is a T-node, the stack or list is cleared.

After this procedure, it is possible that the stacks are not empty, but the tree traversal has already been completed. In this case, the sequence of metrics lying in the stacks will be unloaded into a special file (dump). When retrying decoding after failure, it is possible to use PMF information from the stacks "saved" from previous attempts to more accurately traverse the tree and generate a new list of candidates with the condition of the calculated metrics. This caching allows for more accurate decoding and higher quality performance when running SCL-Creeper several times on the same code. A similar concept is bit-flipping decoding, where iterativeness also has a positive effect on performance. The authors will try to develop the idea of a caching algorithm in future works.

Let ν_c be the root of coding tree, μ_c is related metric, stacks $\mathcal{T} = \mathcal{N} = \emptyset$. List $\mathcal{L}$ has size L, $T = \lfloor \log_2 (N/L) \rfloor$, $\mu_{max} = -\infty$. Next, there is a description of SCL-Creeper algorithm with parameter L in Algorithm 1.

As can be seen from the algorithm, the list is first filled and basic SCL decoding occurs if the path metrics are critically large. However, after going through about half the levels of the code tree, a main list of candidates is formed for which another metric will be calculated - PMF, the calculation and analysis of which uses stacks of valid nodes. Next, when the list overflows, the least reliable paths are excluded (as in the main SCL algorithm). If the list is not full, the stack is cleared.

Each time 2 new paths are added according to the number of "child" vertices at each level. At each iteration, only the most reliable paths and values from the stack are retained. Thus, for one possible path in the tree, it is necessary to store two stacks – for all T-nodes, as well as for unreliable paths for which backward movement is available, similar to SCS. Thus, the overall computational complexity of such an algorithm approach is $\mathcal{O}(LN)$, and the spatial complexity approaches $\mathcal{O}\left(\widehat{D}LN\right)$, where $\widehat{D}$ is the total dimension of the two storage stacks nodes.

Algorithm 1 SCL-Creeper Algorithm

Input: $\mathbf{y}_0^{N-1}$ – received LLRs, L – list size, T – threshold;
(List filling) Find child nodes for ν_c, denote it as ν_1 and ν_2. If $\nu_1 = \nu_2 = \varnothing$ then choose word candidate $\overline{\mathbf{d}}_0^{N-1} \in \mathcal{L}$ (CRC-procedure) and return it;
if $\nu_1 \in \mathcal{F}$ *or* $\nu_2 \in \mathcal{F}$ **then**
  Append 0 to $\mathcal{L}, \mathcal{T}, \mathcal{N}$;
else
  Calculate related PMF metrics μ_1, μ_2 and PM metrics: $\widehat{\mu}_1, \widehat{\mu}_2$;
if $\max(\mu_1, \mu_2) < \min(\widehat{\mu}_1, \widehat{\mu}_2)$ **then**
  Continue tree traversal with SCL algorithm and append ν_1 or ν_2 bit to the candidates' paths from list $\mathcal{L}$;
(Stack filling) Assume that $\mu_1 > \mu_2$. Calculate metric value: $\mu_{cur} = max(Pos(\mathcal{N}),\ \mu_1)$;
if $\mu_{\text{cur}} \geq T$ **then**
  push μ_{cur} to $\mathcal{N}$;
if $\mu_{\text{cur}} > \mu_{\max}$ **then**
  $\mu_{\max} = \mu_1$ and push μ_2 to $\mathcal{T}$;
(List erasing) if $V(\mathcal{N}) \in \mathcal{T}$ **then**
  Pop the least reliable paths from $\mathcal{L} while |\mathcal{L}| \geq L$
else
  (Stack erasing) Pop element from $\mathcal{N}$ stack and $\mathcal{T}$ stack.

VI. SIMULATION RESULTS

e The authors now provide a performance comparison, testing the hypothesis that the list variant will have a much better frame error rate (FER) for different code lengths. One frame contains 10^4 different data blocks with sizes 2^{10} bits per block. Therefore, the comparison was carried out both for the classical algorithms (SC, SCL ($L = 8$), SCS ($L = 8,\ D = 10$)) and for the SC-Creeper ($T = 1$) and SC-Fano ($T = 1$) "convolutional-like" algorithms. For SCL-Creeper we assume that threshold factor $T = \lfloor \log_2(N/L) \rfloor$.

Normalized complexity was also considered and compared for different algorithms. Normalized complexity was chosen as a target analysis tool due to its ease of use: by taking the number of operations required to implement the SC algorithm (which, generally speaking, does not depend on E_b/N_0) and taking this number as 1, a user can build dependencies regarding the complexity of SC. Thus, if an algorithm with a performance higher than that of an SC decoder is still close to SC in terms of normalized complexity, then we can draw a conclusion about its effectiveness.

To simulate the real channel authors use binary phase shift keying (BPSK) modulation and additive white gaussian noise (AWGN) model [16]. The following polynomial was chosen as the basic polynomial for simulation: $g(x) = x^{16} + x^{15} + x^2 + 1$. CRC code with 16 bits provides a higher level of error detection compared to shorter codes. This is important in polar decoding because it helps identify and correct errors more effectively. Moreover, it is a common choice in many communication standards and protocols. 5G NR communications system [6] was chosen for (N, K)-code construction with ratio $R = \frac{K}{N}$ and operations counting module for C++17. The following environment was used to conduct the experiments: Linux distribution (CentOS 8), CPU with Intel Xeon Gold 6230N, RAM with 8*32G DDR4 ECC.

There were obtained the results for performance and normalized complexity of different polar codes with codeword's

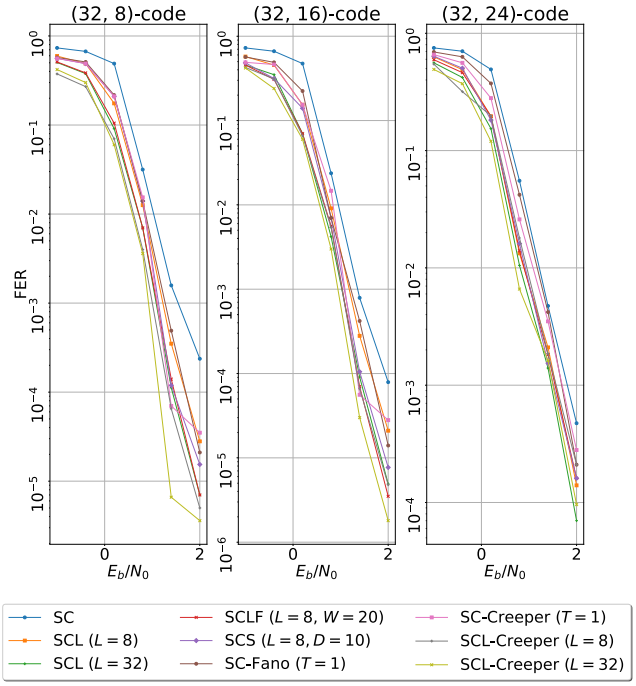

FIGURE 1. SCL-Creeper (list size 8 and 32) performance comparison with other decoders, $N = 32$.

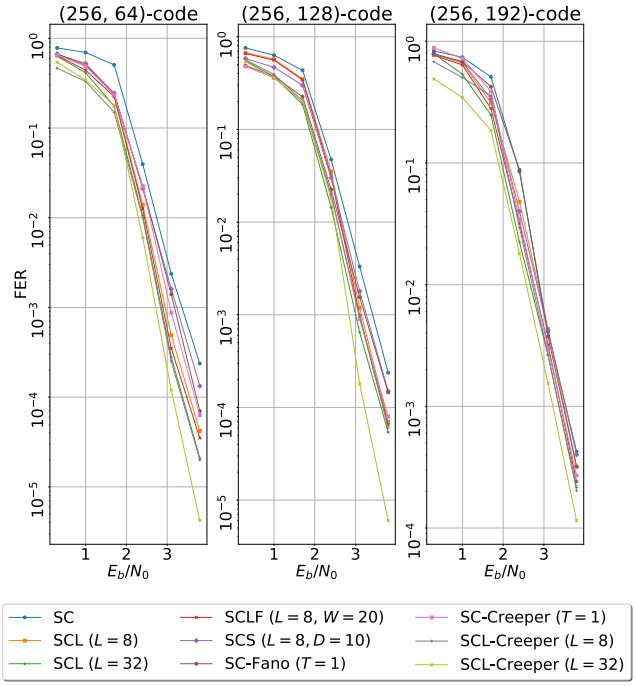

FIGURE 2. SCL-Creeper (list size 8 and 32) performance comparison with other decoders, $N = 256$.

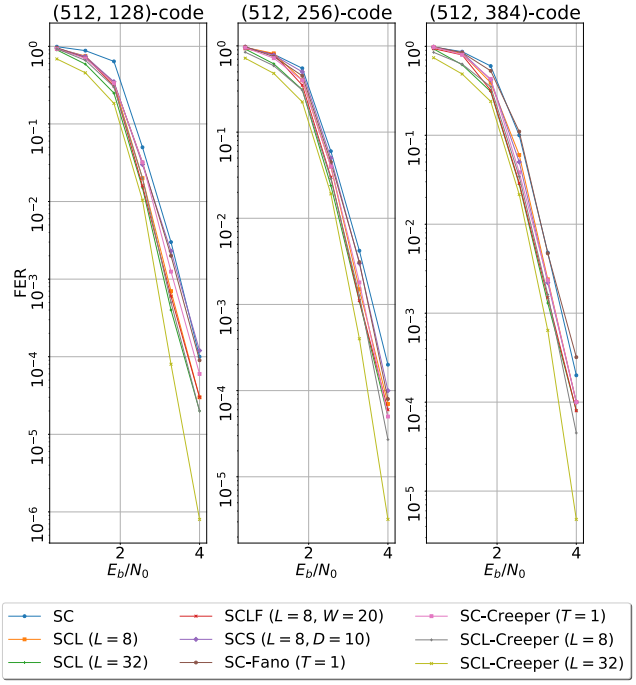

FIGURE 3. SCL-Creeper (list size 8 and 32) performance comparison with other decoders, $N = 512$.

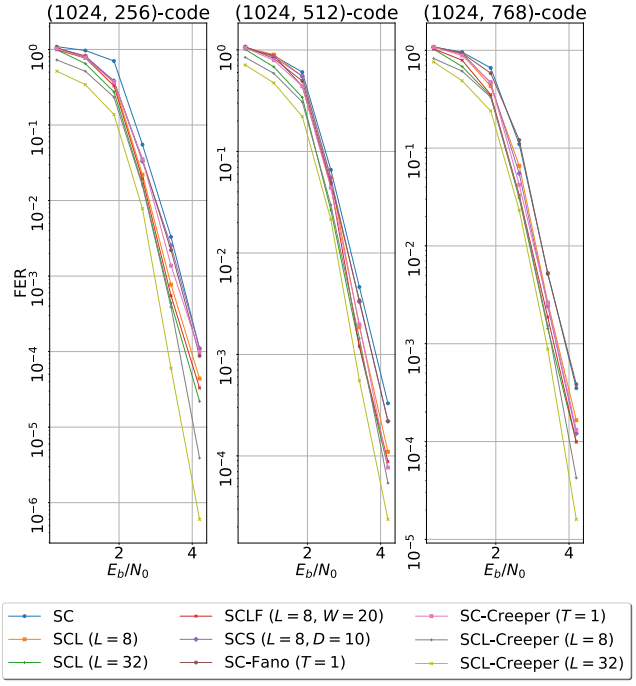

FIGURE 4. SCL-Creeper (list size 8 and 32) performance comparison with other decoders, $N = 1024$.

lengths $N = \{32, 256, 512, 1024\}$: low code rate (1/4), medium code rate (1/2) and high rate (3/4).

The Figures 1-4 show graphs comparing the performance of various decoders, and the Figures 5-8 show how the normalized complexity changes with increasing E_b/N_0 value, respectively. In comparing various polar decoding algorithms several key observations emerge.

SC stands as the baseline algorithm, characterized by its simplicity and relatively low complexity. While it generally performs adequately, it may struggle when faced with higher error rates. SCL algorithms with different list size offer improvements by maintaining lists of candidate codewords. While SCL-32 provides better error correction capabilities, it comes at the expense of increased complexity.

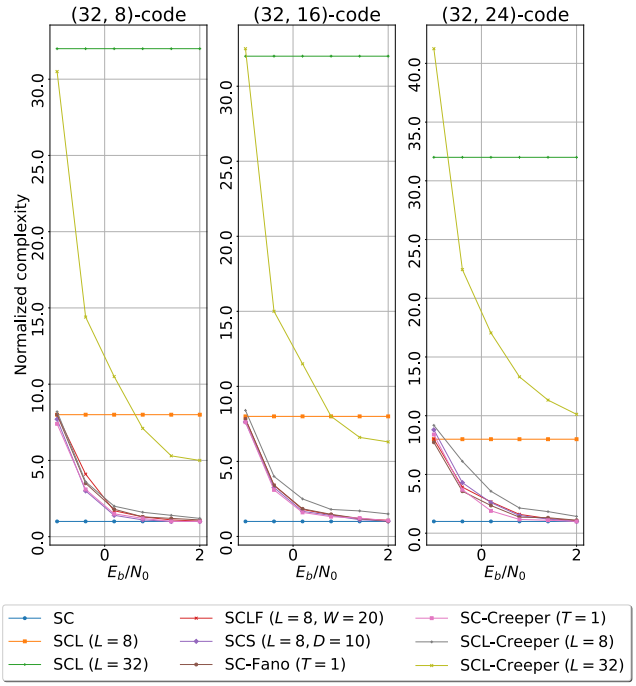

FIGURE 5. SCL-Creeper (list size 8 and 32) complexity comparison with other decoders, $N = 32$.

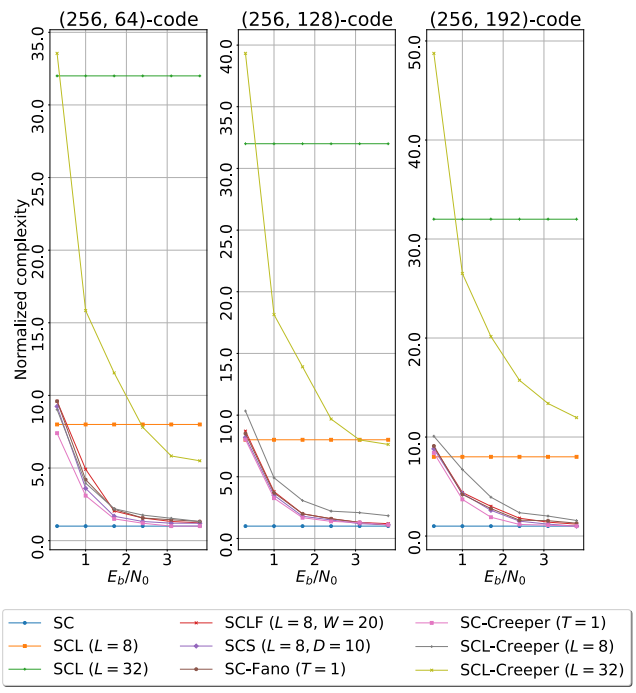

FIGURE 6. SCL-Creeper (list size 8 and 32) complexity comparison with other decoders, $N = 256$.

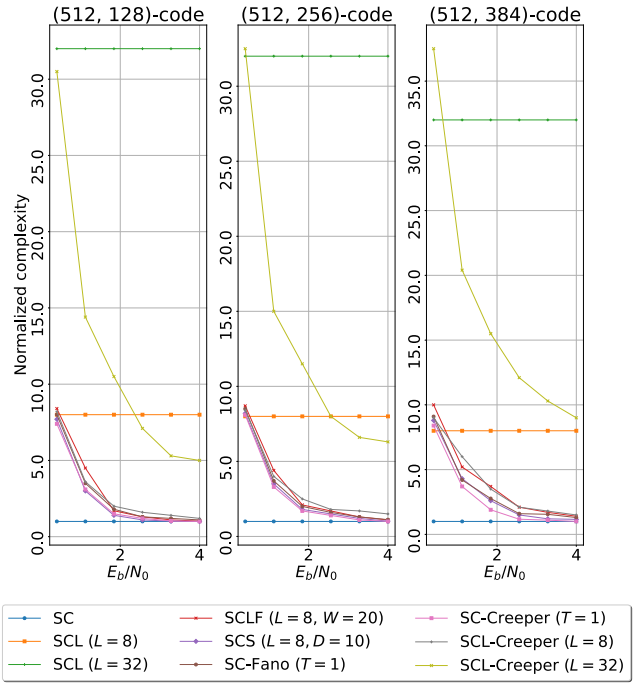

FIGURE 7. SCL-Creeper (list size 8 and 32) complexity comparison with other decoders, $N = 512$.

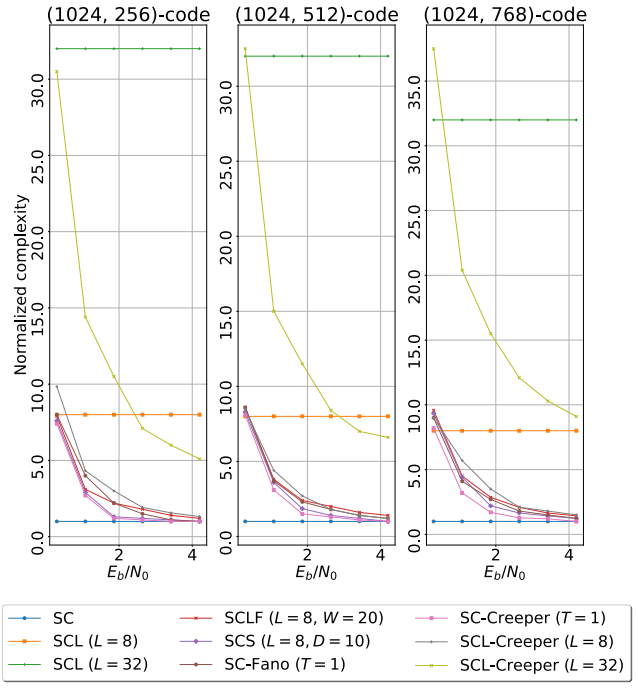

FIGURE 8. SCL-Creeper (list size 8 and 32) complexity comparison with other decoders, $N = 1024$.

Note also that SCLF has good potential for comparison with the SCL-Creeper algorithm: recent studies [22] show that with increased performance in the case of a flipping strategy, the time complexity indicator can also be reduced. However, from Figures 1-4 provided in this work, it can be seen that the use of additional attempts is a good alternative to the Fano method when decoding and allows you to achieve high performance rates.

Further advancements are seen in algorithms like SC-Fano, SC-Creeper, and SCL-Creeper. Notably, SCL-Creeper achieve SCL-32 performance with reduced list sizes, particularly with a list size of 8. This reduction in complexity makes them attractive options for resource-constrained

scenarios. However, it's worth noting that SCL-Creeper may exhibit inferior performance for low code lengths compared to other algorithms due to its reliance on list decoding. Nonetheless, with a list size of 32, SCL-Creeper demonstrates superior complexity compared to conventional SCL-32, making it advantageous in certain environments. Additionally, it benefits from fast convergence with increasing signal-noise ratio values, maintaining good performance without excessive complexity.

Table 1 shows performance gain (E_b/N_0) in comparison with classical SCL approach and SCL-Creeper scheme. It is interesting to note that the difference in E_b/N_0 between SCL-8 and SCL-32 in different simulation scenarios ranges from 0.3 to 1 dB. Thus, based on Table 1 results, we can clearly state that SCL-Creeper with a list length of $L = 8$ achieves the performance of the SCL-8 decoder.

TABLE 1. Performance gain for SCL-Creeper with list sizes 8 and 32.

(N,K)-code	FER	SCL-Creeper vs. SCL ($L = 8$)	SCL-Creeper vs. SCL ($L = 32$)
(256,64)	10^{-2}	+0.13 dB	+0.16 dB
	10^{-3}	+0.23 dB	+0.37 dB
	10^{-4}	+0.52 dB	+0.70 dB
(256,128)	10^{-2}	+0.15 dB	+0.22 dB
	10^{-3}	+0.21 dB	+0.37 dB
	10^{-4}	+0.41 dB	+0.66 dB
(256,192)	10^{-2}	+0.10 dB	+0.19 dB
	10^{-3}	+0.15 dB	+0.30 dB
	10^{-4}	+0.32 dB	+0.43 dB
(512,128)	10^{-2}	+0.17 dB	+0.19 dB
	10^{-3}	+0.25 dB	+0.41 dB
	10^{-4}	+0.53 dB	+0.72 dB
(512,256)	10^{-2}	+0.14 dB	+0.20 dB
	10^{-3}	+0.19 dB	+0.35 dB
	10^{-4}	+0.42 dB	+0.64 dB
(512,384)	10^{-2}	+0.05 dB	+0.17 dB
	10^{-3}	+0.14 dB	+0.27 dB
	10^{-4}	+0.31 dB	+0.43 dB
(1024,256)	10^{-2}	+0.23 dB	+0.22 dB
	10^{-3}	+0.27 dB	+0.42 dB
	10^{-4}	+0.53 dB	+0.70 dB
(1024,512)	10^{-2}	+0.14 dB	+0.21 dB
	10^{-3}	+0.17 dB	+0.33 dB
	10^{-4}	+0.39 dB	+0.61 dB
(1024,768)	10^{-2}	+0.10 dB	+0.15 dB
	10^{-3}	+0.12 dB	+0.23 dB
	10^{-4}	+0.25 dB	+0.37 dB

Now let's fix the normaized complexity value for SC-decoder as 1. Table 2 shows the relationship between the normalized complexity of the classical SCL algorithm and the SCL-Creeper algorithm. It is worth noting that the normalized complexity of SCL is considered to be a constant L, therefore complexity reduction for SCL-Creeper is understood as the ratio of L to the current complexity of SCL-Creeper with list length L.

Next, consider the throughput metric [17], [18], which estimates the peak value of the optimal rate at different code rates. Each throughput value is a multiplication of the current R by $1 - \text{FER}$ for the current decoding option with fixed $E_b/N_0 \equiv 2$ value. For better visualization, the graph was smoothed using spline interpolation. Figures 9-10 show that the optimal value of R for most algorithms is a

TABLE 2. Complexity reduction for SCL-Creeper with list sizes 8 and 32.

(N,K)-code	FER	SCL-Creeper vs. SCL ($L = 8$)	SCL-Creeper vs. SCL ($L = 32$)
(256,64)	10^{-2}	4.5	4.1
	10^{-3}	5.3	5.4
	10^{-4}	6.0	5.8
(256,128)	10^{-2}	3.6	3.3
	10^{-3}	3.8	4.0
	10^{-4}	4.3	4.2
(256,192)	10^{-2}	3.4	2.0
	10^{-3}	3.9	2.3
	10^{-4}	5.1	2.7
(512,128)	10^{-2}	4.0	3.0
	10^{-3}	5.1	4.5
	10^{-4}	5.7	5.5
(512,256)	10^{-2}	3.2	2.9
	10^{-3}	3.9	3.7
	10^{-4}	4.2	4.7
(512,384)	10^{-2}	3.2	1.9
	10^{-3}	3.7	2.1
	10^{-4}	3.9	3.2
(1024,256)	10^{-2}	2.8	2.9
	10^{-3}	3.5	3.9
	10^{-4}	3.7	4.3
(1024,512)	10^{-2}	2.6	2.5
	10^{-3}	3.3	3.7
	10^{-4}	3.5	4.2
(1024,768)	10^{-2}	2.4	2.3
	10^{-3}	2.7	2.7
	10^{-4}	3.1	3.1

parameter approaching 0.8, but for SC and Fano decoders this value is approximately 0.5. It is noteworthy that the highest throughput is achieved by SCL-Creeper with a list length of 32, since the basic SCL ($L = 32$) on average has a lower FER at these code lengths.

As one can see from the Figures and Tables, the SCL-Creeper is indeed capable of better performance than the SCL-8 and the basic SC-Creeper. It is especially interesting that path refinement in SCL-Creeper for short codes turns out to be much more efficient than one-way tree traversal in the case of SCL. Also for short codes it is clear that SCL-Creeper with a list length of 32 at large values of E_b/N_0 FER is much lower than for SCL-Creeper with $L = 8$.

In the general case, using SCL-Creeper with long lists (of the order of 32) turns out to be an ineffective solution, since with a large increase in computational complexity the performance gain turns out to be insignificant.

VII. LIMITATIONS AND DISCUSSIONS

It is obvious that despite the full potential of using the Creeper approach, the development of the idea of improved decoder leads to a certain list of limitations that can be encountered at the synthesis stage with list and stack technologies.

First, the most obvious limitation is memory usage. Since SCL-Creeper uses a list and two stacks, the memory costs are significantly higher than both SCL and SCS decoders. When comparing with them, we understand that the normalized mutiplication factor for SCS is a stack size parameter D (similarly L for SCL), but in our case this factor can reach a value of D^2L. If in the worst case we assume that $D = LN$, then the space complexity of the algorithm grows to $\mathcal{O}(L^3N^2)$. This complexity will be a significant overhead for

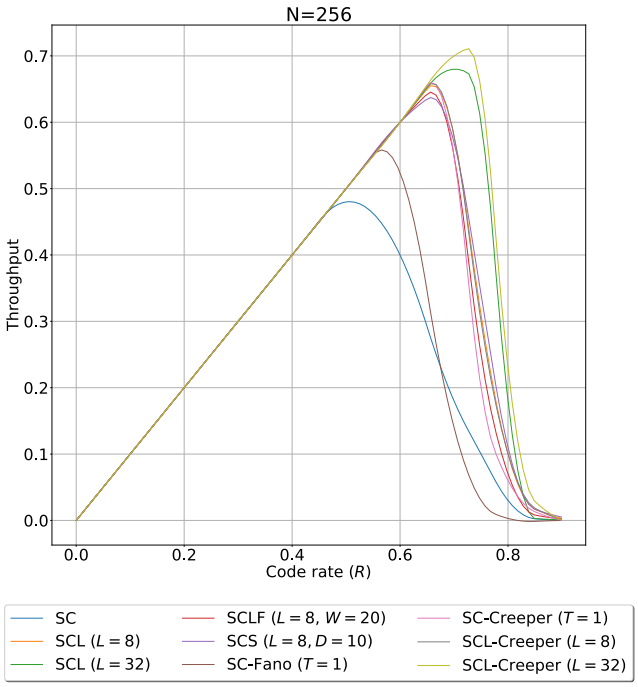

FIGURE 9. SCL-Creeper (list size 8 and 32) throughput comparison with other decoders, $N = 256$.

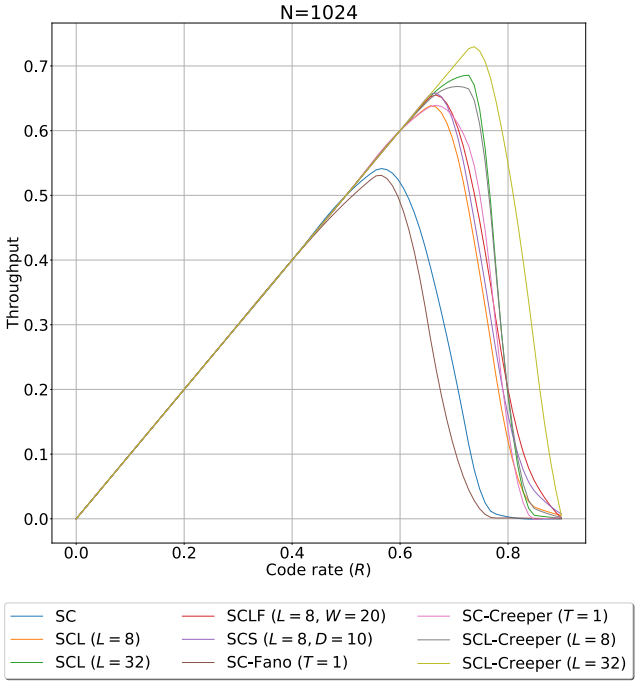

FIGURE 10. SCL-Creeper (list size 8 and 32) throughput comparison with other decoders, $N = 1024$.

medium and large list lengths, as well as for large N codeword lengths.

Next, we should mention the simplification in the current work associated with the inclusion of the threshold parameter T in the heuristic formula. Achieving optimal performance for a given pair of input parameters (L, T) seems to be a non-trivial task within the framework of Creeper decoding, so a similar restriction was chosen for the threshold. Based on the value of the T parameter, the current T-nodes are formed, which affects the progress of the entire algorithm. Therefore, finding a value at which the tree traversal is fast and points to the correct (from the point of view of the original data block) nodes is a future direction for research.

Also, the performance of the SCS-based algorithm may be sensitive to the characteristics of the communication channel, such as noise distribution and fading effects. In scenarios where the channel conditions deviate significantly from the assumptions made during algorithm design, the decoding performance of SCL-Creeper may suffer.

Architecture-dependent improvements are also an important aspect in decoder research. Despite the fairly general description of the SCL-Creeper algorithm, existing research on the hardware applicability of polar decoders and various computational topologies can be used [13], [14]. Since this algorithm represents a fairly high-load method of storing data, optimizing access to it and interaction (send/receive data block) will reduce latency and increase the performance of calculations even with high space complexity. This work has already put forward ideas for caching data from stacks — in this case, by repeating the SCL-Creeper procedure, an accurate result (in a probabilistic sense) can be achieved much faster.

Code tree preprocessing using neural network methods is also a well-known [15] polar decoding technique. Typically, such methods use a combination of "guessing" the most accurate traversal path using a deep learning algorithm, and then applying SCL to traverse that path.

This tactic reduces the complexity of SCL because it avoids unnecessary computation and storage of additional vertices. In the case of the SCL-Creeper algorithm, such preprocessing will help not only to significantly optimize stack sizes, but also to more accurately determine the optimal threshold for calculating T-nodes.

VIII. FUTURE WORK DIRECTIONS

The versatility of the hybrid SCL-Creeper approach opens avenues for further exploration and refinement. By leveraging a diverse array of decoding techniques, including stack, list, and CRC, researchers and practitioners can fine-tune the decoder to meet the specific requirements of different communication environments.

The development of the SCL-Creeper idea is memory optimization (using the reliability matrix [10] as one of the basic modifications) and the use of alternative metrics [11], [12] (not related to the Fano metric in the case of a stack). Of course, the study of additional metrics and improvements in SC-Creeper needs to be transferred to the case of a list decoder.

One of the most promising future directions is the use of flipping strategy in the Creeper algorithm using lists. Since SCF-Creeper has already been described in previous works (see [3], [8]), the main difficulty lies in the synthesis of the list Creeper approach and the flipping strategy. To do

this, additional possibilities for creating a critical set should be explored, because in the SCF-Creeper version there are several options for carrying out the flipping procedure. We should also discuss the computational complexity of such a method, which uses both additional decoding attempts and two lists. The closest reference is the SCLF method, previously discussed in this article. However, a potential SCLF-Creeper algorithm would require a careful balance between extremely high complexity and performance.

Also, cost functions for the Creeper approach are of interest, since you can use more flexible (in terms of the number of operations) methods of traversing the tree than PM/PMF-values. There is a modification to T-nodes that allows traversal to be even faster, but it is not applicable in the current version of SCL-Creeper. Generally speaking, exploring different metrics and dynamic ways to update thresholds will help avoid unnecessary operations and additional passes through the tree in the future.

IX. CONCLUSION

This paper elucidates a captivating narrative of algorithmic evolution, tracing the trajectory of the Creeper algorithm from its origins in convolutional code decoding to its integration within polar codes alongside list-based decoding methodology. The development of this hybrid approach, exemplified by the SCL-Creeper decoder, marks a significant milestone in decoding optimization within digital communication systems. By concurrently employing list and stack techniques, the algorithm showcases a nuanced understanding of decoding dynamics, offering a holistic solution that capitalizes on the strengths of each method.

Despite the augmented spatial complexity inherent in storing $2L$ stacks of comparable lengths (L), the algorithm manages its memory requirements effectively, ensuring that the computational overhead remains reasonable. Moreover, the simplicity of stack operations underscores the algorithm's efficiency, enabling it to navigate the decoding process with precision and agility. This balance between complexity and computational efficacy positions the SCL-Creeper decoder as a compelling solution for real-world deployment scenarios, where reliability and resource efficiency are paramount considerations.

Architectural features of data transmission and quick access to such data are a priority, since SCL-Creeper is the most optimal performance algorithm of all those considered in this work. Combined decoding techniques made it possible not only to achieve minimum FER at low E_b/N_0 at almost all code lengths, but also to maintain the computational complexity of the modified algorithm at the level of the basic SCL decoder. The development of this approach will allow not only to synthesize the ideas of various specialists in the field of polar decoding, but also to arrive at a fundamentally new decoding method at the junction of convolutional and polar codes.

In wireless communication systems, where reliability and spectral efficiency are paramount, the SCL-Creeper algorithm offers significant benefits. By providing robust error correction capabilities while maintaining reasonable computational complexity, it enables more reliable data transmission over wireless channels prone to interference, fading, and noise. SCL-Creeper algorithm's adaptability and performance make it a compelling choice for next-generation communication networks and cloud data storages, where it can enhance the reliability and efficiency of data transmission.

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] H. Aurora, C. Condo, and W. J. Gross, "Low-complexity software stack decoding of polar codes," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1–5.

[3] F. Ivanov, I. Chetverikov, A. Kreshchuk, and I. Timokhin, "Successive cancellation creeper decoding of polar codes," in *Proc. IEEE Int. Multi-Conf. Eng., Comput. Inf. Sci. (SIBIRCON)*, Yekaterinburg, Russia, Nov. 2022, pp. 60–64.

[4] M.-O. Jeong and S.-N. Hong, "SC-fano decoding of polar codes," *IEEE Access*, vol. 7, pp. 81682–81690, 2019.

[5] J. Nyström, R. Johannesson, and K. Zigangirov, "Creeper—An algorithm for sequential decoding," in *Proc. 6th Joint Swedish-Russian Int. Workshop Inf. Theory*, Mölle, Sweden, 1993, pp. 332–336.

[6] F. Rinaldi, A. Raschellà, and S. Pizzi, "5G NR system design: A concise survey of key features and capabilities," *Wireless Netw.*, vol. 27, no. 8, pp. 5173–5188, Nov. 2021.

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

[8] I. Timokhin and F. Ivanov, "On the improvements of successive cancellation creeper decoding for polar codes," *Digit. Signal Process.*, vol. 137, Jun. 2023, Art. no. 104008.

[9] H. Yang, E. Liang, M. Pan, and R. D. Wesel, "CRC-aided list decoding of convolutional codes in the short blocklength regime," *IEEE Trans. Inf. Theory*, vol. 68, no. 6, pp. 3744–3766, Jun. 2022.

[10] M. Albanese and A. Spalvieri, "Two algorithms for soft-decision decoding of Reed–Solomon codes, with application to multilevel coded modulations," *IEEE Trans. Commun.*, vol. 56, no. 10, pp. 1569–1574, Oct. 2008.

[11] D. Zhang, B. Wu, and K. Niu, "Path metric inherited SCL decoding of multilevel polar-coded systems," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Nanjing, China, Mar. 2021, pp. 1–6.

[12] M. C. Liberatori, L. J. Arnone, J. C. Moreira, and P. G. Farrell, "Soft distance metric decoding of polar codes," in *Cryptography and Coding* (Lecture Notes in Computer Science), vol. 9496. Cham, Switzerland: Springer, 2015.

[13] A. Balatsoukas-Stimming, A. J. Raymond, W. J. Gross, and A. Burg, "Hardware architecture for list successive cancellation decoding of polar codes," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 8, pp. 609–613, Aug. 2014.

[14] X. Han, R. Liu, Z. Liu, and L. Zhao, "Successive-cancellation list decoder of polar codes based on GPU," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Chengdu, China, Dec. 2017, pp. 2065–2070.

[15] Z. Ibrahim and Y. Fahmy, "Enhanced learning for recurrent neural network-based polar decoder," in *Proc. 13th Int. Conf. Electr. Eng. (ICEENG)*, Cairo, Egypt, Mar. 2022, pp. 105–109.

[16] H. Li and J. Yuan, "A practical construction method for polar codes in AWGN channels," in *Proc. IEEE Tencon Spring*, Sydney, NSW, Australia, Apr. 2013, pp. 223–226.

[17] F. Cheng, A. Liu, J. Ren, and K. Feng, "CRC-aided parity-check polar coding," *IEEE Access*, vol. 7, pp. 155574–155583, 2019.

[18] H. Khoshnevis, I. Marsland, and H. Yanikomeroglu, "Throughput-based design for polar-coded modulation," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1770–1782, Mar. 2019.

[19] Y. Wang, L. Chen, Q. Wang, Y. Zhang, and Z. Xing, "Algorithm and architecture for path metric aided bit-flipping decoding of polar codes," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–6.

[20] Y. Yongrun, P. Zhiwen, L. Nan, and Y. Xiaohu, "Successive cancellation list bit-flip decoder for polar codes," in *Proc. 10th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2018, pp. 1–6.
[21] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Proc. IEEE ACSSC*, Nov. 2014, pp. 2116–2120.
[22] W. Zhang and X. Wu, "Low-latency SCL bit-flipping decoding of polar codes," in *Proc. IEEE Int. Conf. Commun.*, Rome, Italy, May 2023, pp. 132–135.
[23] Y. Jiang and L. Zhang, "A Fano decoding for polar codes based on node acceleration," in *Proc. IEEE 98th Veh. Technol. Conf. (VTC-Fall)*, Hong Kong, Oct. 2023, pp. 1–5.
[24] Z. Feng and L. Liu, "On the adaptive fano-SC-Flip decoding of polar codes," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Foshan, China, Aug. 2022, pp. 226–231.
[25] S. P. Badar, K. Khanchandani, and P. Wankhede, "Fast polar decoder implementation using special nodes," in *Proc. 2nd Int. Conf. Paradigm Shifts Commun. Embedded Syst., Mach. Learn. Signal Process. (PCEMS)*, Nagpur, India, Apr. 2023, pp. 1–6.
[26] H. Khoshnevis, C. Cao, D. Chang, and C. Li, "Novel design of irregular polar codes for latency reduction in fast polar decoders," in *Proc. IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, Ottawa, ON, Canada, Sep. 2021, pp. 1–5.
[27] Y. Zhao, Z. Yin, Z. Wu, and M. Xu, "Minimum-combinations set-based rate-1 decoder for fast list decoding of polar codes," *IEEE Commun. Lett.*, vol. 25, no. 10, pp. 3185–3189, Oct. 2021.
[28] S. Han and J. Ha, "Polar codes for fast converging belief-propagation decoding," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Jeju Island, South Korea, Oct. 2021, pp. 773–775.
[29] I. S. Timokhin and F. I. Ivanov, "Fast bit-flipping decoding of polar codes with additional nodes," in *Proc. IEEE 24th Int. Conf. Young Professionals Electron Devices Mater. (EDM)*, Novosibirsk, Russia, Jun. 2023, pp. 360–365.

ILYA TIMOKHIN (Member, IEEE) is currently pursuing the master's degree in information security with the Department of Cyber-Physical Systems Information Security, National Research University Higher School of Economics. He is with Huawei's Moscow Research Center, analyzing big data, complex query execution efficiency, and computing resource optimization. He also provides research with the Laboratory of Cyber-Physical Systems and the Internet of Things. His research interests include artificial intelligence, hash function research, polar coding, and graph theory.

FEDOR IVANOV (Member, IEEE) received the M.S. degree in mathematics from Far Eastern Federal University, Vladivostok, Russia, in 2011, and the Ph.D. degree in computer engineering and theoretical informatics from Moscow Institute of Physics and Technologies, Moscow, Russia, in 2014. Since 2011, he has been a Fellow Researcher with the Institute for Information Transmission Problems, Russian Academy of Science. He is currently an Associate Professor with the Department of Electronic Engineering, Department of Cyber-Physical Systems Information Security, National Research University Higher School of Economics, Moscow. His research interests include communication theory, polar codes, LDPC codes, concatenated codes, convolutional codes, and non-orthogonal multiple access.