Overview of Various Methods for Decoding and Constructing Critical Sets of Polar Codes

Ilya Timokhin and Fedor Ivanov

***Abstract*—Polar codes have gained significant attention in recent years as they offer a promising solution for reliable communication in the presence of channel noise. However, decoding these codes remains a critical challenge, particularly for practical implementations. Traditional decoding methods such as belief propagation and successive cancellation suffer from complexity and performance issues. To address these challenges, authors have researched several low-complexity decoding techniques, including bit-flipping decoding with critical set construction. Bit-flipping decoding methods operate by flipping a limited number of bits in the received codeword to bring the decoder output closer to the transmitted message. The critical set construction is an essential component of these methods, which identifies the set of bits to be flipped. This paper compares various bit-flipping decoding methods with different critical set constructions, including revised critical set, subblocks-based critical set, key set and others. The performance of these methods is evaluated in terms of bit error rate, computational complexity, and an average number of operations. In summary, this paper provides a comprehensive overview of bit-flipping decoding methods with critical set construction for polar codes. The paper's findings highlight the potential of these methods to improve the performance and reliability of polar codes, making them a viable option for practical implementation in modern communication systems.**

***Index Terms*—Bit-flipping, critical set, decoding, hardware implementation, heuristics, polar codes, SC.**

I. Introduction

POLAR codes, introduced by Arıkan [1], are known to be capacity-achieving codes that can provide significant error correction performance, but their decoding is a computationally challenging task. Thus, authors are looking for new strategies and approaches to construct a decoding basis that provides optimal polar decoding, which requires the development of efficient decoding algorithms, path metrics, data structures, and critical sets [27]. Decoding performance can suffer for finite block lengths when using successive cancellation (SC) decoding, which is optimal for infinite lengths. To overcome this issue, SC decoding can be extended by using a list of candidate codewords, leading to the successive cancellation with a list (SCL) algorithm [2]. A further improvement can be achieved by using cyclic redundancy check (CRC) codes to verify candidate codewords in the list. The CRC-aided SCL (CA-SCL) algorithm [3] adds a CRC code to the end of the list candidates and checks whether the resulting codeword is valid. This additional step can improve error correction performance, leading to a lower decoding complexity [7]. Moreover, some special types of nodes in the polar code tree can be decoded without additional processing steps and with lower complexity than SC decoding. These nodes are called special and can be decoded using the generalized SCL (GSCL) method [35], [36].

The basic decoding methods, including SC, SCL, and GSCL, have high space complexity, which can be addressed by using bit-flipping strategies [8], [9]. These methods use the same code tree and decoding algorithms but with an iterative approach and additional attempts to decode a sequence by flipping some bits. This is motivated by the fact that SC-based methods may fail to give the correct output due to a single error during decoding. Bit-flipping strategies include shifted-pruning [20], [21], dynamical [28], [29], generalized [10] and other methods.

Different decoding strategies exhibit varying advantages, such as high error correction performance and lower computational complexity. The study of critical sets architecture and their applicability allows [19], [59] to significantly reduce the spatial complexity of SCL [64] or SC [65], but for non-critical growth of computational complexity it is necessary to reduce the number of additional decoding attempts. This is achieved by accurate selecting a critical set.

Currently, several investigations are underway on popular methods of polar decoding. In [11], basic methods of polar construction and their applicability to additive white Gaussian noise (AWGN) channel are considered. This paper focuses solely on code construction methods and doesn't consider any decoding method. Another paper [12] describes hardware metrics and implementation processes for SC and SCL decoders across different areas, supplies, channels, and technologies of hardware application. However, it only analyzes basic methods of decoding, and the dependency between different values of code length and signal-to-noise ratio (SNR) is not analyzed, with only a constant code rate of 0.5 and code length of 1024 bits presented. On the other hand, there are a number of studies comparing the implementation of the scheduling and flipping methods for specific systems [23]–[25]. An overview of decoding methods with varying parameters for various code lengths has been presented in [13]. This paper also provides hardware implementation features and power consumption

Manuscript received January 9, 2023; revised August 30, 2023; approved for publication by Tarable, Alberto, Division 2 Editor, September. 26, 2023.

I. Timokhin and F. Ivanov are with National Research University Higher School of Economics Tikhonov Moscow Institute of Electronics and Mathematics Ringgold standard institution, Department of Electronics Engineering, Moskva 123458, Tallinskaya st., 34, Moscow, Russian Federation, email: doublemind21@gmail.com, fivanov@hse.ru.

The study was implemented in the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) in 2023.

I. Timokhin is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2023.000049

1229-2370/23/$10.00 © 2023 KICS

analysis. However, it doesn't compare flipping-based methods with other approaches. A paper [14] comparing different polar decoding methods with turbo codes and LDPC approaches provides a strong rationale for the comparison methodology and differences between such coding methods. The book [15] presents various basic decoding methods, their analysis, scope, and variations, including detailed characteristics of special nodes for the generalized method.

Various optimizations and special designs are also presented [50], [51] in the works on code construction design and performance enhancement in decoded sequences. Many optimizations related to building a fast decoding approach [53], [54] are also outside the scope of this study, but the effectiveness of many of them has influenced [58], [63] the development of flipping methods with critical sets using optimization data. The possibility of redistribution and efficient usage of resources in the decoding process also remains outside the scope of this work, but is described in detail in [60]. It is interesting to note that for polar codes, such classical algorithms from the theory of information, such as spherical decoding [61], [62], syndromic decoding [57], Fano-like decoding [52], [56]. An optimization similar to bit flipping is the rewind technology [26], which allows you to recalculate the least reliable bits in reliability sequences. This method allows to significantly save memory by storing partial information about the code word, which has a positive effect on the complexity characteristics of the decoder. The efficiency of all described methods is also quite high, as they are recommended in many communication systems.

II. Common Notation

Since this work contains information about various decoding methods, critical set construction and algorithms, the reader may get confused in the notation. For this purpose, the following notation will be used hereinafter:

- $\mathcal{A}$: The information set of polar code bits;
- $\mathcal{A}^c$: The frozen set of polar code bits;
- N: Number of bits in polar codeword (length of the encoder's output);
- K: Number of bits in the message without frozen bits;
- v_x^i: Common notation for a slice of the vector of length $i - x - 1$ from x position to i position. If $i = \varnothing$ then it describes the xth bit of the vector v.
- $\mathbf{d}$: Initial message which consist of information and frozen bits;
- $\mathbf{u}$: The codeword after modulation.
- $\widehat{u}$: The output word of polar decoder which should be compared with $\mathbf{d}$;
- $\mathbf{y}$: Input LLRs for decoder after encoding and modulation;
- $\alpha_m^{(x,i)}$: mth vertex of polar tree on the level x, where $i = \{l, r\}$, l means the left child vertex in tree and r means the right child vertex respectively. If $i = \varnothing$ then it describes the vertex in current level only. If $m = \varnothing$ then it describes the whole level, but not the specific vertex;
- L, R, U, f, g: Additional functions to evaluate output LLRs in SC-decoding scheme;
- PM_x^i: ith path metrics for list-based decoding variance on xth iteration of calculation.
- $\mathcal{U}$: Critical set for flipping-based decoding strategies;
- n': Length of the special node in generalized methods;
- $\mathrm{PM}_{\mathrm{SPC}}$, $\mathrm{PM}_{\mathrm{G\text{-}Node}}$, $\mathrm{PM}_{\mathrm{G\text{-}Rep}}$: path metric values for each type of the special node;
- M: Vector of the alternative metrics instead of path's or belief tree's metrics.
- σ: Approximation factor for oracle-based critical set;
- β: Threshold factor for key set;
- η: Normalization factor for normalized critical set;
- P_e: The frame error rate (FER) expectation function based on erfc function which is a lower bound for critical set's metrics.

III. Preliminaries: Polar Encoding

Polar codes utilize the channel polarization phenomenon to achieve reliable communication over noisy channels. This is achieved by applying the polar transformation procedure, which strictly divides subchannels into two groups based on the presence of noise, yielding either noisy or noise-free channels. It involves the systematic transformation of information bits into codewords that can be reliably transmitted over a noisy channel. This section provides a detailed overview of the polar encoding process, from code construction to information mapping.

The construction of polar codes begins with the identification of reliable and unreliable channels through channel polarization. Given a binary-input memoryless channel, the Bhattacharyya parameter is used to quantify the reliability of each channel. Channels with lower Bhattacharyya parameters are deemed more reliable and are utilized for information transmission.

A key insight lies in the fact that polarization of channels can be achieved by recursively combining smaller instances of the base channel. This process, known as channel combining, results in a hierarchy of channels that exhibit varying degrees of polarization. The channels with the highest reliability are referred to as "frozen" channels, and the less reliable ones are referred to as "unfrozen" channels. Each channel's polarization provides the ability to split the input bits into the information set $\mathcal{A}$ and the frozen set $\mathcal{A}^c$.

The encoding algorithm for polar codes capitalizes on the polarization phenomenon to systematically map information bits onto the frozen and unfrozen channels. The steps involved in the encoding process are as follows:

1) Initialization: Given a set of information bits to be transmitted, initialize the code block and determine the positions of frozen and unfrozen channels based on their Bhattacharyya parameters.

2) Channel splitting: Divide the code block into smaller sub-blocks through recursive channel splitting. Each sub-block consists of a pair of channels, where one channel inherits the reliability of the original channel and the other channel inherits the unreliability.

3) Channel flipping: For each sub-block, apply channel flipping to transform the unreliable channel into its comple-

ment. This operation ensures that the more reliable channel is used for information transmission.

4) Bit mapping: Map the information bits onto the frozen channels while leaving the unfrozen channels unchanged. This mapping is performed based on the desired encoding rate and the specific polar code construction.

A (N, K)-code means that K information bits and $N - K$ frozen bits are chosen, where $N = 2^n$ and $n \geq 0$. The process of polar encoding can be represented as the following equation:

$$\mathbf{u} = \mathbf{d}\mathbf{G}_N. \tag{1}$$

In (1), the codeword $\mathbf{u} = (u_0, \cdots, u_{N-1})$ is linearly received after the $\mathbf{G}$ transformation of the message $\mathbf{d} = (d_0, \cdots, d_{N-1})$, where $\mathbf{G}_N = F^{\oplus n}$, $n = \log N$, and the $\oplus n$ operation means the n-fold Kronecker product of $\mathbf{F}$.

The polar encoding process begins with the initialization of each bit to obtain the vector of encoded bits $\widehat{u}_0^{N-1}$. This procedure can be represented as a tree with $\lambda = \log N$ levels and a Plotkin construction to construct each vertex on the upper level from two child bits.

Polar decoding is used to enhance the error correction performance of the received codeword vector u_0^{N-1}. The optimization objective of such decoders is to maximize the throughput while minimizing the decoding channel's latency.

IV. Classical Decoding Methods

In this section, we consider several strategies and concrete algorithmic implementations of decoding methods for polar codes. All of them provide different complexities, reliabilities, and performances, but the main goal of each decoder is to find the exact codeword after exposure to Gaussian noise and distortion of the encoding message.

A. Successive Cancellation Decoder

The upcoming chapter discusses the intricacies of the SC decoding technique, a foundational method for polar decoding. SC decoding employs a pragmatic strategy to traverse the coding tree in search of the message vector value. This procedure revolves around three critical operations: The L-step, the R-step, and the encoding step.

In the context of SC decoding, we'll introduce a set of notations to elucidate the process more effectively:

$f(a, b)$: An auxiliary function used in the L-step, defined as $f(a, b) = \text{sign}(a)\text{sign}(b)\max(|a|, |b|)$.

$g(a, b, c)$: Another auxiliary function used in the R-step, defined as $g(a, b, c) = b + (1 - 2c)a$.

The L-step operation essentially involves moving from the vertex $\alpha^{(t)}$ at level $\lambda = t$ to the left vertex $\alpha^{(t-1,l)}$ at the preceding level. This movement is encapsulated by the function L, which utilizes the auxiliary function $f(a, b)$.

The R-step procedure occurs subsequent to the L-step and involves transitioning from the right vertex $\alpha^{(t,r)}$ to the vertex $\alpha^{(t+1)}$ on the same level. This transition is guided by the function R, employing the auxiliary function $g(a, b, c)$.

Furthermore, we have the U-step, commonly referred to as the encoding step: It is responsible for recalculating the value of $\alpha^{(t+1)}$ using the known values of $\alpha^{(t,l)}$ and $\alpha^{(t,r)}$, which are obtained after performing the L-step and the R-step, respectively.

By employing these operations—the L-step, the R-step, and the U-step—SC decoding navigates the polar coding tree in a systematic manner, allowing for the determination of the message vector's value. This approach forms the cornerstone of polar decoding and serves as a foundational concept in modern coding theory.

After the recursive execution of the L, R, and U steps, the SC decoder obtains a vector α^0. Each element (weight) of the LLR output vector corresponds to the expected message bit from $\widehat{u}_0^{N-1}$ by the following rule:

$$\widehat{u_i} = \begin{cases} 0, & i \in \mathcal{A}^c \text{ or } \alpha_i^0 \geq 0, \\ 1, & \text{otherwise}, \end{cases} \tag{2}$$

where $\mathcal{A}^c$ is the complement of the set $\mathcal{A}$ that contains the indices of the information bits.

Consider a simple polar code with a block length of $N = 8$, $\mathcal{A} = \{0, 4, 6, 7\}$ and a design parameter $R = 1/2$, meaning half of the bits will be used for transmitting information. The polar code construction uses a specific transformation matrix to determine which bits are transformed into information bits and which become frozen bits.

We start with the received noisy codeword and initialize the input log-likelihood ratios (LLRs) vector y.

We begin at the top of the coding tree with $\alpha^{(0)}$, which is the root node. We have two child nodes: $\alpha^{(1,l)}$ (left child) and $\alpha^{(1,r)}$ (right child). For each bit, we calculate the new LLR values using the L-step: $y_{1,0}^{(1,l)} = f(y_1^{(0)}, y_2^{(0)})$, $y_{1,1}^{(1,l)} = f(y_3^{(0)}, y_4^{(0)})$, $y_{1,2}^{(1,l)} = f(y_5^{(0)}, y_6^{(0)})$, $y_{1,3}^{(1,l)} = f(y_7^{(0)}, y_8^{(0)})$.

Now we perform the R-step, which uses the parent node values and the already updated LLRs from the L-step. For each bit, we calculate the new LLR values using the R-step: $y_{1,0}^{(1,r)} = g(y_1^{(0)}, y_2^{(0)}, y_{1,0}^{(1,l)})$, $y_{1,1}^{(1,r)} = g(y_3^{(0)}, y_4^{(0)}, y_{1,1}^{(1,l)})$, $y_{1,2}^{(1,r)} = g(y_5^{(0)}, y_6^{(0)}, y_{1,2}^{(1,l)})$, $y_{1,3}^{(1,r)} = g(y_7^{(0)}, y_8^{(0)}, y_{1,3}^{(1,l)})$.

Finally, we update the parent node $\alpha^{(1)}$ using the updated child node values from both L-step and R-step. We continue this process iteratively, moving down the tree level by level. At each level, we calculate LLRs for both left and right child nodes, update the parent node using these values, and move to the next level. This iterative process continues until we reach the leaf nodes of the tree, which correspond to the original information bits. Once we have LLR values for the leaf nodes, we can make decisions about the transmitted bits. For example, if the LLR is positive, we might decide that the transmitted bit $\widehat{u}_i$ is "1" and if the LLR is negative, we might decide that the transmitted bit is "0".

The SC decoding algorithm can achieve symmetric channel capacity, but its main disadvantage is that it is unreliable, and it does not allow for backtracking on previous steps in the graph if the codeword contains errors. Many techniques and methods have been proposed [39], [40] to improve the reliability of SC decoding, such as hardware parallelization and hyperthreading for the code tree, which have been shown to give higher performance for the SC case.

B. Successive Cancellation List Method

The SCL decoding method employs a similar traversal over the coding tree as the SC method, but instead of outputting the final decoded message, it creates a list of possible message vector candidates, and the last step involves selecting the correct candidate from this list.

To achieve this, an extra layer (sublayer $\lambda = -1$) is added, and both 0 and 1 vertices are assigned for each non-frozen bit. The likelihood vertex is chosen based on a special path metric (PM) that is calculated recursively. Note that the parameter L represents the cardinality of the list of possible candidates.

The PM metric is defined as follows:

$$\mathrm{PM}_i^t = \begin{cases} \mathrm{PM}_{i-1}^t + |\alpha^{(t)}(i)|, \ \widehat{u}_i \neq \frac{1-\operatorname{sign}\alpha^{(t)}(i)}{2}, \\ \mathrm{PM}_{i-1}^t, \ \text{otherwise.} \end{cases} \tag{3}$$

Here, $t = 1, \cdots, L$ is an index of the related candidate's path, and the initial value for the path metric is $PM_{-1}^t = 0$. The output of this method is not the correct codeword but rather L most likely candidates. For instance, for $L = 4$, it returns four candidates with the highest likelihood for the codeword.

Let's consider an example from previous subsection on the SCL decoder with $L = 4$.

For each candidate path $t = \{1, 2, 3, 4\}$, compute the PM values. Begin with $PM_{-1}^t = 0$ for all paths. For each bit i, calculate PM_i^t based on the (3) and obtain $\alpha^{(t)}(i)$ value from SC decoding U-step.

Starting from level 0 and moving down the tree, update the PM values for each candidate path based on the recursive formula. At the leaf level, for each candidate path, calculate the final PM value. Select the candidate paths with the highest PM values to create the list of potential message vector candidates.

There are various approaches for selecting the correct candidate from the list of possible candidates. Some methods use distance metrics, while others employ machine learning techniques (see [5]). However, one of the most well-known and straightforward approaches is to use CRC values (see [6]). This involves adding a special sequence at the end of the message, and decoding results are verified using a cyclic detector that checks the validity of the obtained algorithm answer.

The CA-SCL decoding scheme is a specific type of SCL decoder that outputs the SCL candidate paths into a CRC detector, and the check results are used to detect the correct codeword.

C. Generalized Decoding Method

Define **special nodes** [34], [53] as the set of nodes in the tree representation that can be decoded with much less complexity than provided by the basic SC method. The main idea is to use decoding not sequentially (bit by bit) but through the pattern of special nodes presented below (see [22], [31] for different patterns descriptions and analysis):

- "Frozen" node: Contains only frozen bits, and the code consists of only one trivial codeword of zeros;
- "Information" node: Contains a vector of ones, and the code consists of all possible vectors;
- REP-node (repetition): All bits (except the right-most) are frozen, related codewords are ones or zeros vectors;
- SPC-node (single parity-check [32]): All bits (except the left-most) are information, codewords have only even Hamming weight [33];
- Generalized repetition node (G-Rep): Any node at level t for which all its descendants are "frozen" nodes, except the rightmost one at a certain stage $p < t$, which is a **generic node** (G-Node);
- Generalized parity-check node (G-PC): Any node at level t for which all its descendants are "information" nodes, except the leftmost one at a certain stage $p < t$.

Rules are used to decode the special nodes in non-trivial cases (only the "frozen" node is trivial).

The "information" node can be decoded as follows: If it is placed in positions $k = i, \cdots, i + n' - 1$ (n' is the length of the special node) in some codeword, then the most likely codeword $\widehat{u}_i^{i+n'-1}$ has the following structure according to the input LLR vector $\mathbf{y}$:

$$\widehat{u}_k = \begin{cases} 0, & \mathbf{y}_k > 0, \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

It is decoded by splitting it into two parts, and then calculating and sorting the related PM metrics. If the critical set $\mathcal{U}$ does not contain indices from i to $i+n'-1$, then the path from $L+1$ to $2L$ will be removed from the list. Otherwise, the first L paths will be eliminated.

SPC-nodes decoding provides the same idea. Let's assume it is placed on the same k positions from i to $i + n' - 1$. The procedure for initializing the codeword is as follows:

$$\widehat{u}_k = \begin{cases} 0, \ \text{if } i \in \mathcal{A}, \\ 1, \ \text{otherwise.} \end{cases} \tag{5}$$

The proposed GSCL method allows for providing computations only for the SCL metrics at the top of the coding tree. In such an iterative way, we can obtain new path metrics for generalized nodes and provide fast polar decoding. The process of special node decoding is described in detail in the paper [31].

D. Flipping Decoding Methods

1) Basic flipping method: The successive cancellation flip (SCF) method described in [8] 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.

Assuming that CRC checking occurs after the termination of SC decoding, if the CRC method fails (which means that the validated sequence does not match the predetermined CRC from the message), then the SC-Flip method tries to swap bits according to critical set values (see the next section to learn about different schemes of critical set construction). A detailed description of the SCF decoder is presented in Algorithm 1.

Algorithm 1 SCF algorithm

Set the parameter T for additional attempts;
Receive an output likelihood vector $\widehat{u}_0^{N-1}$ from the basic SC-decoder;
if $T > 1$ and CRC checking for $\widehat{u}_0^{N-1}$ fails **then**
  for $j = \overline{1, T}$ **do**
    Calculate (update) the critical set $\mathcal{U}$;
    Invoke SC-decoding for received vector and flip the value of $\widehat{u}_{\mathcal{U}_j}$;
    Receive a new possible codeword $\widehat{u}_0^{N-1}$ and related metrics;
    if CRC checking validates $\widehat{u}_0^{N-1}$ **then**
      Return $\widehat{u}_0^{N-1}$ as the correct codeword.
    end if
  end for
end if
Return $\widehat{u}_0^{N-1}$.

If $T = 0$, the algorithm has no attempts to recalculate CRC, and it is equivalent to basic SC decoding. In essence, the SCF approach introduces a strategy for iteratively refining the results of SC decoding through controlled bit-flipping and validation, thereby offering improved error correction performance. This technique demonstrates its effectiveness particularly in scenarios where noisy channels or other uncertainties demand multiple iterations to discern the correct codeword.

This algorithm has a significant improvement in performance compared to the SC-decoder, but its complexity is poor because it invokes the SC algorithm T times. There are some improvements to the basic SCF strategy, which are described below.

2) List flipping decoder: 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. In [9], it is indicated that the noise properties of the vector $\mathbf{y}$ are not taken into account after the construction of the critical set (denoted as $\mathcal{U}$ as before).

This algorithm also uses the parameter T, as in the SCF approach, which represents the maximum number of bit flip attempts. Typically, $T \approx |\mathcal{U}|$. 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 $\widehat{u}_1^N$ vector. It initializes the $\mathcal{U}$ set, and for each attempt $i \in T$, it tries to flip the ϕth bit in the jth path from the set of paths. Each time it can flip only one bit and try CRC validation via the SCL method.

This method achieves slightly improved performance for the SCF approach [46], 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.

3) Dynamical flipping decoder: This method uses the oracle critical set and a simple modification of the SCF approach to make it valid for both SCF and SCLF decoding methods.

The dynamic SCF algorithm differs from the approaches considered earlier in the ability to use bit-flipping for several bits at once. Thus, the critical set uses a level parameter that determines the number of bits required for flipping (for example, a 2-level critical set allows flipping the two least "reliable" bits of the sequence in one iteration).

Also, DSCF uses a metric that affects the tree trajectory, which allows you to quickly find the most bit positions that do not need to be changed when flipping again. Thus, during the first iterations, the most "reliable" bits are determined, for which the bit-flipping procedure is either not needed at all, or is needed only once, and at further iterations, the group of those bits that the algorithm has marked as the least "reliable" are refined. The dynamic feature of this decoder also lies in the fact that each subsequent attempt strictly depends on the previous results of decoding and bit-flipping.

Thus, the probability of finding a correct initial sequence does not decrease with each new iteration of the algorithm. For example, SCF cannot guarantee this accuracy due to the static bit-flipping procedure and situations where new flipping attempts can degrade the FER. Also, despite attempts to store a group of bits in memory instead of a single bit, the algorithm provides a theoretical complexity comparable to the SCF approach, increasing the probability of no error due to a heuristic method for constructing a critical set and a metric for choosing a group of flipping bits from [28] (Algorithm 3).

It is also worth noting the practical recommendations announced by the developers of this algorithm. So, for example, to reduce the complexity of the decoder, it is proposed to update not by all values of the critical set, but by several indices, where the metric value reaches the largest (or smallest, depending on the code design) value. For DSCF, equivalent SC decoding options are also defined, in which the values of the output LLR will match.

For the SCLF decoder, there is also a dynamic modification to recalculate indices, and its structure is similar to the DSCF decoder. However, the critical set updating procedure can be described as follows: We choose the smallest $T - t - 1$ values from the set $\tilde{\mathcal{U}} = \mathcal{U}_{t+1}, \cdots, \mathcal{U}_{T-1}$ and update the related indices of the bit-flipping set with such smallest values: $\mathcal{U}_{t+1}, \cdots, \mathcal{U}_{T-1} = \tilde{\mathcal{U}}_1, \cdots, \tilde{\mathcal{U}}_{T-t-1}$.

4) Generalized flipping decoder: The paper [10] presents a novel approach that combines list-based decoding with bit-flip decisions. This approach improves computational efficiency and provides higher performance compared to the SCLF decoder. Additionally, it generalizes some node classes and decodes them efficiently. The resulting algorithm is called generalized SCL with flipping (GSCLF). The critical set $\mathcal{U}$ is constructed using a list-based approach.

The GSCLF method uses generalized principles for constructing special nodes. For example, the second most likely codeword $\widehat{u}_0^{i+n'-1}$ from the "information" node flips the bit $\widehat{u}_r$ where r is the index that minimizes $|\mathbf{y}_r|$ among the values $i, \cdots, N-1$.

It should be noted that the authors of the current algorithm use the conventional Fast-SCL decoding approach [35]. Firstly, $\mathbf{y}_0^{N-1}$ is divided into several special nodes. Then for each special node (except in the "frozen" case), PM metrics are

calculated. If any of the CRC codes are validated successfully, the most likely codeword with the minimum index over all suitable codewords is returned. Otherwise, the critical set $\mathcal{U}$ is constructed according to (7).

5) Shifted-pruning flipping method: In SCL decoding, the list only stores L paths with the highest PM values, but the correct codeword path may be lost and eliminated from SCL (SCLF) due to channel noise. This method [20] keeps the lowest metrics in the critical set and modifies the removing approach (authors call it the novel **shifted pruning** method) in additional decoding attempts.

It modifies the range of selecting the paths: For SCL, SCLF, and DSCLF decoding, it chooses the first L correct paths according to their metrics, but the shifted-pruning method works with paths from $k+1$ to $k+L$ for some k.

A shifted pruning scheme is proposed to significantly reduce the probability of eliminating the correct path in additional decoding attempts. The critical set heuristics is a basic variant for this method, but it could be simply changed to another complex set with different path metric calculations.

6) Other flipping decoding methods: This paper only observes the most basic and comparable algorithms of bit-flipping decoding, but there are many other approaches and models of decoders with flipping ability. Let's describe (initially) some interesting decoding methods with bit-flipping ability in the following bullets.

- *Partitioned SCF*: In [41], the authors propose the partitioned SC-Flip (PSCF) algorithm, which divides the code sequence into different partitions based on additional parameters and updates the critical set according to the probability of failed decoding. PSCF corrects at least a single wrong decision with a probabilistic approach and maximizes the probability of successful decoding of a single bit. However, this method does not provide good performance metrics because it uses only the error per bit entropy without the common probability of the codeword.
- *Neural methods*: Other approaches [38], [42] are strongly based on neural networks and oracle algorithms, using imitation learning and convolutional neural networks to provide higher decoding performance. Such methods require high competence in machine learning science and have high complexity due to the code tree updating procedure after each refinement of neural parameters. The presence of a large number of computational operations such as aggregation, stratification, and normalization makes it difficult to test and analyze these approaches compared to simpler and more straightforward decoding methods.
- *Layered flipping method*: A method described in [18] involves dividing the code tree into subblocks and adding labels for each level. After each attempt, a new critical set is generated with blocks (subtrees), and an optimal way to correct as many errors as the current level allows is determined. The authors note that this method is comparable to the CA-SCL ($L = 4$) approach and achieves the best performance compared to the basic SC algorithm.
- *SNR-based decoder*: This paper [45] is based on an approximation of LLR thresholds and SNR dependency for flipping methods. It uses a generalized approach for special node decoding and heuristics calculations for bit-flipping decoding decisions. In other words, the authors found a correlation between SNR values (generally speaking, code ratio) and the accuracy of the GSCF method. This decoding approach uses a threshold to dynamically regulate the current probability of error and to calculate criteria for updating the critical set.
- *ISFSCL*: Memory reduction is an important feature for polar decoding, and an improved segmented flipped SCL (ISFSCL) decoder [44] has been presented. It uses the segmented approach described in [43] and analyzes the basic SC decoder (especially its binary tree construction methods) according to the critical set. This method provides a uniform distribution of codeword blocks for additional memory optimization. However, its performance is comparable only with SCL with a list capability of $L = 2$ but provides lower complexity due to optimizations with code segments and tree splitting. Another segmented approach [47] uses improvements for CRC-checking and the oracle algorithm. It provides higher performance compared to the ISFCSL algorithm (due to the novel generalized scheme and CRC refining), but requires a significant amount of memory.
- *Belief propagation method*: Another probability-based algorithm is based on belief propagation decoding [16]. This algorithm is heuristics-like and strongly based on some parametric improvements for the conventional belief propagation decoder, which is similar to the basic SC decoder. It provides a flexible multilevel update for likelihood bits but uses several matrices to store error probability ratios, which is a significant disadvantage compared to an SCL approach.
- *Key set SCLF*: It's interesting to note that there is a scheme with no critical set usage [48]. The authors presented the key set with some memory improvements (including the bits which are easy to eliminate from the list). Another memory addressing method is applicable for several system architectures, but this approach cannot provide lower complexity for the common case. The key set is constructed from special node states and operates with its indices with the suggestion that access to these indices can be faster than construction and updates for the critical set.

V. Principals and Strategies of Critical Sets Constriction

In general, flipping methods employ classical approaches, such as SC, SCL and GSCL decoders, to calculate path metrics and determine critical sets in order to improve their accuracy and redefine LLR values. These decoders aim to enhance the error correction performance of the basic SC method, which suffers from the problem of insufficiently accurate search for errors. The flipping methods try another suggestion (depending on the bit from $\widehat{u}_0^{N-1}$) to address this problem.

This section is divided into two parts: Critical set construction and word decoding using different strategies.

A. Construction of Critical Sets

Flipping methods rely on the concept of the critical set, which is a set of indices that need to be flipped. The critical set $\mathcal{U}$ stores path metrics, which are used to recalculate bits on the LLR.

As mentioned earlier, different bit-flipping strategies use various methods to construct the critical set. However, it is important to distinguish between the evaluation of the critical set and the concrete bit-flipping decoder. In other words, we can consider the problem of the applicability of several critical sets for different decoders. Let us describe some methods for designing a critical set and whether it is possible to construct this set using the decoders described earlier.

- **Naive critical set:** [8]: The naive set $\mathcal{U}$ consists of the T smallest absolute (path's or belief tree's) metric values ($|\mathcal{U}| = T$). This critical set flips only one bit per attempt, so we can call it a 1-level set.
- **Revised critical set:** [9]: The revised critical set (RCS) is a novel critical set applicable for list-based decoders. Naive critical set does not take into account the noise design (e.g., the frozen bits) in the input LLRs values, which would affect its accuracy. To overcome this issue, revised critical set was developed.
  We denote **SC state** as such a state if, in some path during the decoding of bit $\widehat{u}_i$, a situation arises when one of the child nodes is preserved after sorting the array among the L candidates, and the second node is not preserved. **CD state** (clone or deletion) denotes a state in which the path is not in SC state.
  If L (list length) is a power of 2, then $\mathcal{B}_1 \in \widehat{u}_0^{N-1}$ describes the first $\log L$ bits from the received codeword which contains only information positions. Let $\mathcal{U}$ be a naive critical set, and $\mathcal{B}_2$ denote the set of indices which paths is in CD state.
  The structure of the transformation from naive critical set to revised critical set is described below:
  $$\mathcal{U} = \mathcal{U} \setminus (\mathcal{B}_1 \cup \mathcal{B}_2). \tag{6}$$
  We can prune and mark new nodes with this method, but it has no flexibility or dynamic update. However, this scheme provides us with the ability to flip the whole $\mathcal{U}$ set or choose only one bit per attempt.
- **List-based critical set:** [10]: This method is simpler than the revised critical set's evaluation and doesn't use any special node manipulations. This method requires much less memory, being a difference scheme for a naive critical set, in which the least stable metrics are arranged in such a way that their corresponding bits will be subject to the flipping procedure. Let's assume we have a set of path metrics from the SCL method. The critical set should be constructed as follows:
  $$\mathcal{U} = \{i_1, \cdots, i_T \ : \ M_{i_1} \leq \cdots \leq M_{i_T}\}, \quad M_i = -\mathrm{PM}_i^0 + \mathrm{PM}_i^L. \tag{7}$$
- **Subblock critical set:** [18]: Unlocking a novel avenue for elevating the error correction performance of polar codes, the subblocks critical set technique intricately engages with the subblock structure inherent in the full binary polar tree. Rooted in a noteworthy proposition from [19], this approach surges ahead by capitalizing on the intriguing property that decoding an entire subblock correctly hinges solely on the accurate decoding of its first bit. This striking discovery fuels the approach's foundational principle: A high likelihood that the critical set encompasses the initial error bit, strategically chosen as the bit where bit-flipping maneuvers are most efficacious. Central to this method is the construction of a polar code tree composed of subblocks. Each subblock embodies a distinctive property: if the initial bit of a subblock is correctly decoded, the entire subblock can be confidently decoded. This phenomenon heralds a high potential for accurate error correction through strategically chosen bit-flipping.
  The process of constructing the critical set involves a division of the polar codes into multiple subblocks, each with a coding rate of $R = 1$. Within each subblock, the approach identifies the first unfrozen bit and accumulates these bits to form the set $\mathcal{U}$. This selection strategy aligns with the foundational proposition, as the initial error bit is deemed pivotal for subsequent error correction endeavors. The subblocks critical set technique doesn't halt at bit-level corrections. Instead, it orchestrates multilevel flipping across the entire subtree associated with the chosen subblocks. This orchestrated manipulation of the subtree aligns with the endeavor to achieve accurate error correction for the chosen subblocks.
- **Heuristics critical set:** [20], [21]: This approach, outlined in [20] (Algorithm 1), serves as an intriguing alternative due to its independence from tree or code design considerations. This independence opens doors for the method's integration into a wide array of existing decoders, rendering it versatile and applicable across diverse scenarios. The procedure revolves around generating the critical set $\mathcal{U}$ through a well-defined algorithm, as detailed in [20]. The method's elegance lies in its simplicity, allowing it to stand independently from the intricacies of code construction or tree design. This simplicity paves the way for seamless integration with existing decoders, enabling their augmentation with improved error correction capabilities. Acknowledging the trade-off inherent in this approach, it offers a means of addressing its potential limitations. By modulating the parameter governing extra retries and integrating a more robust decoder, one can strategically enhance error correction outcomes. This fine-tuning not only underscores the method's adaptability but also underscores its compatibility with different coding scenarios.
- **Normalized critical set:** [30]: This method utilizes the same approach as list-based set construction. However, instead of selecting the T smallest path's values, it selects the T smallest M_i metrics from [29], with a normalization factor η that mitigates the biased estimate caused by

propagated errors:

$$M_i = \ln \left[\frac{\sum_{l=0}^{L-1} \exp\left(-\mathrm{PM}_i^l\right)}{\left(\sum_{l=0}^{L-1} \exp\left(-\mathrm{PM}_i^{l+L}\right)\right)^{\eta}} \right]. \quad (8)$$

Assuming we have a set of path metrics values received from the SCL method, we sort it in ascending order and construct a normalized critical set based on the flip metrics as follows:

$$\mathcal{U} = \{\infty, \cdots, \infty, M_{i+\log_2 L}, \cdots, M_{T+\log_2 L}\}, \quad (9)$$

This approach uses normalization instead of absolute metrics, as in the simple list-based construction. However, choosing the correct value for η is crucial in minimizing the error probability. This method allows for dynamic approaches, and flipping metrics can be easily recalculated for each step. Note that this normalization is a generalized case of list-based critical set absolute metrics.

- **Oracle-based critical set:** [28]: This method is similar to the heuristics-based construction strategy but employs additional parameters to prevent errors and provide feedback for updating the critical set.
  In cases where the initial SC decoding step encounters failure, the chosen bit-flips of order 1 correspond to positions $i \in \mathcal{A}$ with the lowest $|y_i|$ values. However, it's important to recognize that using the absolute value of the LLR as a likelihood metric for bit-flip decisions is not the most optimal approach. This is because such a metric fails to incorporate the sequential nature of the SC decoding process.
  In fact, while a lower absolute value of the LLR does indicate a higher likelihood of error in the corresponding hard-decision bit, it does not provide any insight into the probability of that error being the first one occurring during the sequential decoding process. In simpler terms, this metric does not differentiate the initial error from subsequent errors. To address this limitation, the novel oracle metric aimed at evaluating the likelihood of a bit-flip with the purpose of rectifying the trajectory of the SC decoding process.
  By "correcting the trajectory" we refer to the scenario where SC successfully decodes all bits u_i. It's crucial to note that this doesn't imply the overall success of the SC decoding, as there is no guarantee that it will successfully decode subsequent bits.
  In essence, while the SC Flip decoder targets improving error correction by altering bit values based on the LLR, the conventional absolute LLR-based metric doesn't capture the sequential nature of the decoding process. This metric, on the other hand, evaluates the likelihood of a bit-flip with the aim of rectifying the trajectory of the decoding, although it doesn't guarantee overall success.
  Let σ be the approximation factor. This method constructs the set $\mathcal{U}$ iteratively for each propagated error using the output LLR metrics set from the basic SC decoder in the following way:

$$\mathcal{U}_i = \sum_{j \in \mathcal{U}_{i-1}} \mathrm{PM}_j \quad (10)$$

$$+ \frac{1}{\sigma} \sum_{j \in \mathcal{A}: j < \mathcal{U}_{i-1}} \ln(1 + \exp\left(-\sigma \mathrm{PM}_j\right)); \quad (11)$$

- **Key set:** [48]: This method involves using generalized decoding and is based on pre-removing some special nodes from the set. Such an assumption is closely related to memory optimization and access. Instead of a critical set, a key set is used for nodes that can be quickly retrieved from the code tree. In addition to SC state and CD state, which are described in the revised critical set, this approach also employs SC-DEL state nodes that should be removed from the flipping set. The normalized error probability is calculated similarly to that of the critical set and is presented as follows:

$$M_i = \log \frac{\sum_{l=1}^{L} \exp(-\mathrm{PM}_{l+L}^i)}{\sum_{l=1}^{2L} \exp(-\mathrm{PM}_l^i)}. \quad (12)$$

The FER expectation $P_e(i)$ of bit u_i is calculated as follows:

$$P_e(i) = \frac{1}{2} \mathrm{erfc}\left(\frac{\sqrt{\mathrm{y}_i}}{2}\right), \quad (13)$$

where erfc is the complementary error function. The key set $\mathcal{U}$ can now be constructed by combining the u_i indices which is located in CD state and SC state. After that it will remove the indices u_i which is located in SC-DEL state and left only indices which metrics are required the following statement:

$$M_i > \beta \log P_e(i), i \in \mathcal{U}, \quad (14)$$

where β is a threshold factor that reduces the key set's length by limiting the number of SC nodes and fixing the critical set's size.

- **Belief propagation critical set:** [16]: A pioneering technique in refining the SCF decoder's efficacy is the belief propagation critical Set, as expounded in the work by [16]. This approach leverages specialized nodes and their recalculations through classical belief propagation decoding, culminating in a novel perspective on polar code optimization. The foundation of this method is constructed within a factor graph representation of the polar code, thereby lending it a structured and comprehensible framework.
  The factor graph is a graphical model that vividly illustrates the relationships between variables and constraints in a coding scenario. In our context, it encompasses variable nodes (representing bits) and check nodes (corresponding to constraints). This representation unveils the underlying structure of the polar code, aiding in the exploration of potential improvements. The belief propagation process unfolds as follows: Iteratively, the beliefs associated with variable and check nodes are updated. These beliefs capture the probability distributions of bit values and assess their correctness. Iteration by iteration, the belief propagation process refines the beliefs

assigned to variable nodes. As iterations progress, a point of decision arises. Once a predetermined number of iterations is completed or a stopping criterion is satisfied, the decoder crystallizes its conclusions regarding bit values. These conclusions extend to the formation of indices that comprise the critical set $\mathcal{U}$.

The critical set construction methodology is an integral aspect of this approach. After meticulously refining beliefs through the belief propagation iterations, the decoder faces the pivotal task of determining which bits to potentially flip. To achieve this, it sorts the indices of unfrozen bits in descending order of their significance. The top T indices, where T is a parameter governing refinement attempts, are then meticulously chosen to constitute the ordered set. This set of indices encapsulates the bits that hold the most promise for error correction, thereby guiding the SCF decoder in its subsequent bit-flipping endeavors. By integrating belief propagation set construction into the SCF framework, this method taps into the dynamics of variable and check nodes to enhance the decoder's accuracy, making it particularly suited for error-prone scenarios where sophisticated error correction is imperative.

- **Segmented critical set:** [47]: Segmentation is an efficient decoding strategy for SC-based methods (e.g., using subtrees or probabilistic characteristics for certain code segments). We divide the codeword into s different parts. Let M_i be the normalized critical set metrics from (8). Then we recalculate the segmentation metric as follows:

$$M_i = \frac{M_i}{\sum_{j=\lfloor i/s \rfloor}^{\lfloor i/s \rfloor + N/s} M_j}, \tag{15}$$

  then it should be sorted in descending order. The critical set consist of the T largest M_i values.
- **SNR-based critical set:** [49]: An innovative technique aimed at bolstering the error correction capabilities of polar codes hinges on an essential factor in communication channels: The SNR. This SNR-Based Critical Set approach employs a meticulous methodology that harnesses SNR information to construct a critical set in a segmented manner. This strategic segmentation is carried out with an overarching goal of recalculating indices, underpinned by the objective of diminishing FER values. Conversely, for lower SNR values, where the noise effect is more pronounced, additional iterations are engaged to fortify the error correction capabilities. An integral aspect of the SNR-Based Critical Set approach is the identification of segmentation points for the critical set formation. These points delineate boundaries within which the recalculations take place. The allocation of these points, however, is not arbitrary; rather, it is informed by a penalty mechanism intricately tied to the SNR. This penalty serves as a guiding force, ensuring that the recalculations are attuned to the SNR level. This adaptive penalty takes into account the prevailing SNR conditions to achieve a fine-tuned recalibration of the critical set.

TABLE I
LINKAGE BETWEEN DIFFERENT CRITICAL SETS AND DECODERS.

CS	Approach	Base dec.
Naive set	SCF	SC
Revised	GSCLF	SCL
List-based	SCLF, DSCLF	SCL
Subblocks	SCF, Shift.-prun.	SC+SCL
Heuristics	SCF, Shift.-prun.	SC+SCL
Normalized	SCLF, DSCLF, GSCLF	SCL
Oracle-based	SCF, DSCF	SC
Key set	GSCLF	SCL
Belief set	SCF, DSCF, Shift.-prun.	SC
Segmented set	SCF, DSCF	SC
SNR-based set	SCLF, GSCLF	SCL
Neural	SCLF, GSCLF	SCL

  It adapts to the channel's specific characteristics, striving to achieve the most accurate bit decisions while being mindful of the computational resources and time constraints.
- **Neural-based critical sets:** This is an advanced technique that employs neural networks to identify optimal critical sets for error correction in polar codes. Instead of relying on traditional methods, this approach leverages the capabilities of neural networks to learn and predict which bits should be flipped to enhance error correction performance. This method involves training a neural network using labeled training data that pairs received noisy codewords with the corresponding corrected codewords. Once trained, the neural network can predict critical sets for new received codewords, guiding the error correction process more effectively. This technique combines the power of neural networks with polar code error correction, offering a data-driven and efficient way to improve error correction outcomes. Such sets could be obtained via machine learning methods and neural networks LSTM refining [17]. This field is out of scope for the current paper, but neural-based decoders show high error-correction performance and are worthy of mention.

B. Applicability of Critical Sets

Even though decoding methods with predefined critical sets have been described previously, it can be useful for certain decoders to change the method of critical set construction and try to implement another bit-flipping engine.

In Table I, let us consider which decoding method applies to the specific critical set and which basic decoding approach (SC or SCL) can be used in this case.

As we can observe, some methods are only applicable with SC decoder while others are only suitable for SCL decoder. However, certain construction methods can be employed for both SC decoder and list method as well.

The key set has a complex structure, making its evaluation and design challenging, and hence it can be easily applicable only to the generalized list-based decoder. On the other hand, to provide fast first-step calculations, we assume that the key set is based on the list-based critical set, which can be effortlessly computed for SCLF and DSCLF methods. Thus, the key set combines the construction of its initial set from

TABLE II
THEORETICAL COMPLEXITY COMPARISON.

Method	Average compl.	Worst case compl.	Memory compl.
SC	$O(N \log N)$	$O(N \log N)$	$O(N)$
SCL	$O(LN \log N)$	$O(LN \log N)$	$O(LN)$
SCF	$O(\varphi N \log N)$	$O(TN \log N)$	$O(N)$
SCLF	$O(\varphi LN \log N)$	$O(TLN \log N)$	$O(LN)$
GSCLF	$O(\varphi LN \log N)$	$O(TLN \log N)$	$O(LN)$
DSCF	$O(\varphi N \log N)$	$O(TN \log N)$	$O(N^2)$
DSCLF	$O(\varphi LN \log N)$	$O(TLN \log N)$	$O(LNT)$
Shifted-pruning	$O(\varphi LN \log N)$	$O(TLN \log N)$	$O(TL)$

another approach (basic and dynamic list methods) and applies it to the generalized decoder. The belief propagation method can be effortlessly incorporated into all types of list-based decoders. However, the generalization of such an approach entails some overheads for evaluations, and that's why the GSCLF decoder was excluded for such a critical set.

The SNR-based set is not a reliable solution for a dynamic method because it uses a probabilistic regime to decrease possible SNR values and fix them. Thus, there is no need to modify it dynamically.

VI. COMPARATIVE ANALYSIS

We have defined three metrics to compare the different decoding methods:

1) **Error correction performance**: This metric describes the relationship between the simulated Frame Error Ratio (FER) and the SNR (E_b/N_0).
2) **Normalized complexity**: This metric compares the SC-decoder (the baseline solution) to other decoding strategies as the SNR increases.
3) **Average elementary operations**: This metric consists of two classes of operations, C_1 and C_2, which describe the number of multiplications, log-operations, and exp-operations in C_1, and the number of sums, XORs, comparisons, and sign inversions in $\pm 1 \to \mp 1$ in C_2.

First, we present the theoretical complexity values for each decoder in Table II. Here, L represents the list of possible candidates (for SCL, SCLF, and GSCLF), or the path's length constraint (for SCS). T represents the maximum number of attempts to flip the bit for SCF and its modifications, $\varphi \in [1, T]$ is an average number of flips with current SNR-value and critical set construction method.

In this study, we used the basic critical sets for each decoder type, as described in their original papers. We used the 5G NR reliability sequences for polar construction of (N, K)-code with a ratio of $R = K/N$ and an operations counting module for C++17.

For CRC checking in list-based methods (such as CA-SCL), we used a polynomial with 16 additional bits: $g(x) = x^{16} + x^{15} + x^2 + 1$. We applied these CRC prefixes with the 5G NR polar approach described above for SCL, SCLF, DSCLF, and GSCLF schemes for codeword selection. We considered code length $N = 256$ (and $N = 64, 1024$ for elementary operations research) with $R = 1/2$. The chosen 5G NR code

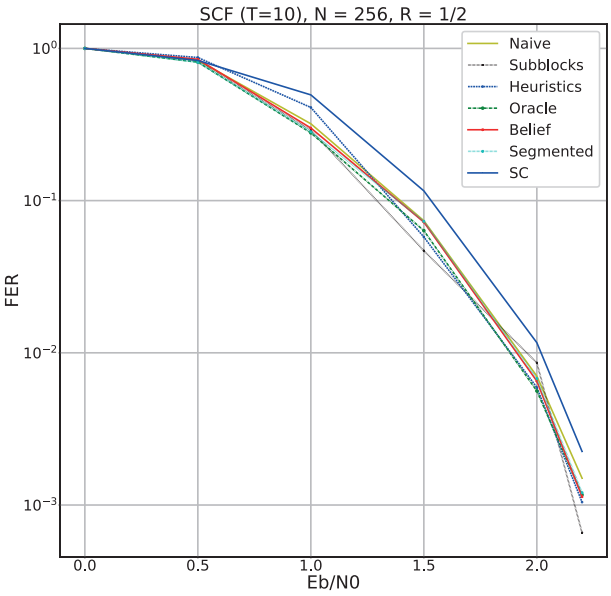

Fig. 1. Performance of critical sets construction for SCF ($T = 10$) decoder.

scheme provides 25, 40, and 50 subcodes for the related code ratios R (excluding subcodes of frozen nodes). Therefore, if we apply bit-flipping methods with extra attempts $T > 20$, it is possible to flip the majority of information bits in the scheme (information general node).

Our simulation shows that for DSCF it's better to use $T/4$ group of bits to flip, that's why we choose 3-level critical set for dynamic flipping algorithm.

We will denote the CA-SCL method as "SCL ($L = x$)", where x is the list size. We choose a classical moderate list size ($L = 8$), and a large list with a high possibility of error correction ($L = 32$). We also set $L = 8$ for other list-based decoding strategies (GSCLF, DSCLF).

For the modulation of 5G NR polar codes, we used the AWGN channel with the binary phase shift keying (BPSK) function [4], as described in [11].

A. Performance Analysis of Critical Sets

In this subsection, we present a comprehensive set of tests for comparative analysis. The experiments are divided into three parts: Investigation of the performance and complexity of different critical sets for each decoding method, analysis of the performance and complexity of different decoders with optimal critical set decision, and evaluation of the number of elementary operations involved.

SCF: Fig. 1 depicts the performance of critical sets construction for the SCF ($T = 10$) decoder. We selected the following critical sets to explore the basic flipping strategy based on the SC decoder: the naive approach and an improved heuristic version of constructing such a set.

The SC decoder shows significant improvement compared to other methods that use 10 bit-flipping attempts. However, the heuristic method is poorly adapted to changing the number T and is not flexible enough to choose the necessary indices of the critical set.

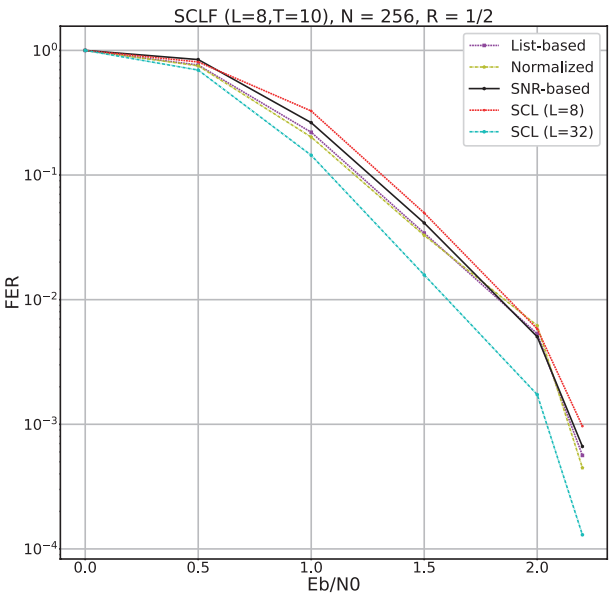

Fig. 2. Performance of critical sets construction for SCLF ($L = 8, T = 10$) decoder.

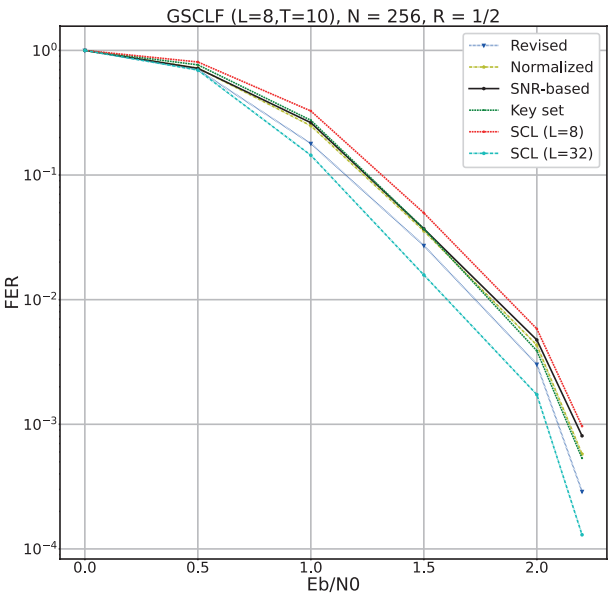

Fig. 3. Performance of critical sets construction for GSCLF ($L = 8, T = 10$) decoder.

The naive critical set performs more efficiently than the heuristic for a short code length, and as the code speed increases, it outperforms not only the heuristic but also more complex methods, such as segmented. The belief propagation approach performs well for all code lengths.

The use of the heuristic method does not yield good performance results. The segmented method provides average performance values at small FERs and low performance values with increasing code length and E_b/N_0. Thus, two similar methods, subblocks and segmentation, yield completely different error correction results. Subblocks division, which is based on a simpler idea of calculating subtrees with $R = 1$, yields a more significant result for a simple decoder. We will see later that the segmentation method itself is not unreliable, but it is not recommended for use with an SCF decoder.

SCLF: In Fig. 2, we present the performance test results for the SCLF decoder. For comparison, we chose the following methods for constructing a critical set: A list filter based on the normalization metric, its simplified version (list-based), and a probabilistic method that selects indices based on theoretical SNR values. The normalized method is a combination of a list-based method and some metric values added to each index of the list-based set. It is worth noting that we implemented the SNR method without the improvements described in the original article. Nevertheless, the idea of performance optimization itself, based on possible SNR indicators according to some probabilistic formula, seemed interesting, motivating us to use the SNR-based critical set in this article. We also chose SCL decoders (without flipping) as a baseline solution for comparison with different list lengths ($L = 8, 32$).

Based on the results, we can conclude that the normalization approach is superior for the SCLF method, as evidenced by its consistently high performance. Notably, for an average code length, the simplified list-based method can even compare with the normalized method. Additionally, there exist specific values for which the SCLF-8 outperforms SCL-32. As for the SNR-based method, it demonstrates comparable performance to the normalized method, indicating a high level of accuracy for this probabilistic approach.

Overall, each of the analyzed algorithms is applicable to the SCLF strategy, but the normalized and simplified list-based methods are the easiest to implement, as they only require the calculation of path metrics without additional computations. In contrast, the SNR-based method utilizes probabilistic calculations to determine the optimal critical set indices and apply the SCLF decoder.

GSCLF: Fig. 3 depicts the basic SCL decoders for two list sizes and several different critical sets. GSCLF is a modified version of the SCL method that relies on the decoding of special nodes. It takes the list-based critical set as a basis and modifies it by introducing some generalized nodes that the algorithm decodes.

The key set based on another version of the critical set is applicable for this method. The construction takes place in two stages: Selection and decoding on a critical set, followed by the construction of a key set based on refinements for the previously considered critical variant. The list-based construction was chosen as the base critical set for the key variant, and this decoding can be considered comparable to the revised method.

The SNR-based algorithm's performance is comparable to the basic SCL-8 approach, which is not favorable for this case at all code rates. The performance of the normalized set and the key set are similar at almost all code speeds. Since the list-based approach is a simplification in terms of the normalized metric, and the key set uses exactly this method, we can conclude that the key improvement did not increase the decoder's overall performance (list-based still fixes fewer errors than normalized, like the original version). The SNR-based critical set can reach the values of the revised critical set.

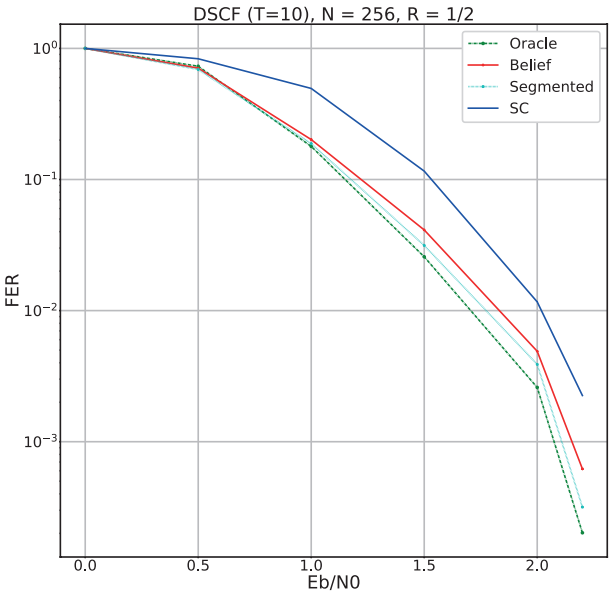

Fig. 4. Performance of critical sets construction for DSCF ($T = 10$) decoder.

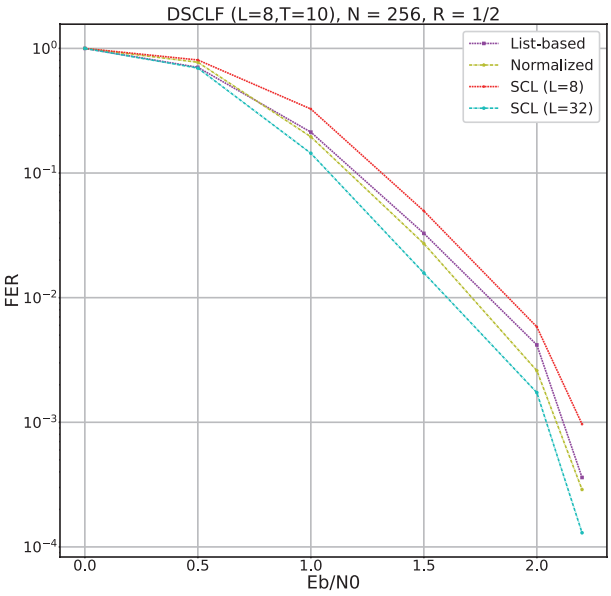

Fig. 5. Performance of critical sets construction for DSCLF ($L = 8, T = 10$) decoder.

The key set method based on the critical one turned out to be rather weak in terms of performance since it did not introduce qualitative changes.

Thus, we can say that the most optimal and adapted method for this algorithm is the revised method, which uses all the features of generalized decoding, providing not only good performance but also low theoretical complexity due to fast decoding. Additionally, the normalized critical set is a simplified approach with similar performance over a large number of decoding attempts.

DSCF: In Fig. 4, an analysis of the performance of the DSCF algorithm is presented with different critical sets, including the oracle-based, belief propagation, and segmented critical sets. Each of these sets has the ability to perform multilevel recalculation of the critical set and refine its indices.

The belief propagation method is observed to perform much lower than the other methods. As the value of E_B/N_0 increases, the performance becomes similar to the SC decoder, which is a suboptimal result. Contrary to theoretical values, belief propagation loses performance as the value of T increases.

On the other hand, the oracle set and segmentation method perform much higher. The oracle-based set performs an order of magnitude higher for large values of E_b/N_0, indicating that the algorithm proposed in the original article is the most efficient in terms of performance. It should also be noted that the DSCF method, due to the use of dynamic permutations and index refinements, provides higher performance than the SCF method, as confirmed by the figures.

The oracle-based heuristic method provides the best applicability for the DSCF method due to the iterative construction method and the detailed analysis of the σ approximation factor described in the original article. However, other critical sets can also be dynamically tuned for multilevel flipping.

In summary, we can consider the DSCF method optimal in terms of performance for decoding without a list (using the basic SC algorithm). However, the trade-off between performance and expended resources for a dynamic algorithm is important to consider, and the impact of dynamic modifications on the decoding complexity requires further study.

DSCLF: In Fig. 5, we can observe the performance of the DSCLF decoder for $L = 8$ and $T = 10$. This value of L allows us to compare the decoder's performance with the basic SCL-8 algorithm and gain insight into how to tune the algorithm to behave like SCL-32 with a dynamic approach and list size of $L = 8$.

The shifted-pruning method employed in the DSCLF algorithm is quite intriguing in terms of parameterization. It does not require the exact number of attempts to iterate T, as this information is calculated heuristically and tied to the critical set's power (i.e., the number of additional attempts).

Based on the foregoing, the authors conclude that the subblocks approach is the optimal method for constructing a critical set in terms of performance. By designing the decoder to rapidly prune those subtrees that do not meet the requirements of a complete binary tree, we can achieve an efficient technique that is strictly dependent on subtrees.

B. Complexity Analysis of Critical Sets

For each of the decoding methods, various critical sets were analyzed, and their normalized complexities were determined based on the complexity of constructing the critical sets for the basic SC and SCL-x decoders. It is noteworthy that complexity normalization was performed to provide a fair comparison between the different methods.

SCF: The normalized complexity values for the basic flip method in SCF are illustrated in Fig. 6. It is evident that the naive method, which employs a 1-level critical set, has the lowest resource requirements since it does not entail any additional resources for its implementation. On the other hand,

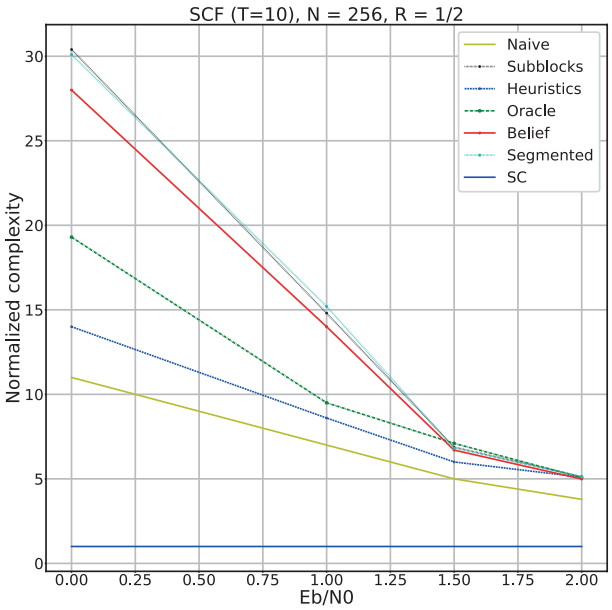

Fig. 6. Complexity of critical sets construction for SCF ($T = 10$) decoder.

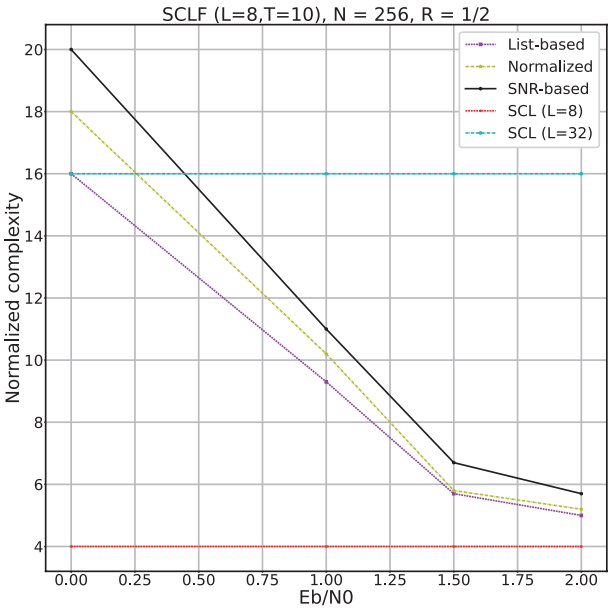

Fig. 7. Complexity of critical sets construction for SCLF ($L = 8, T = 10$) decoder.

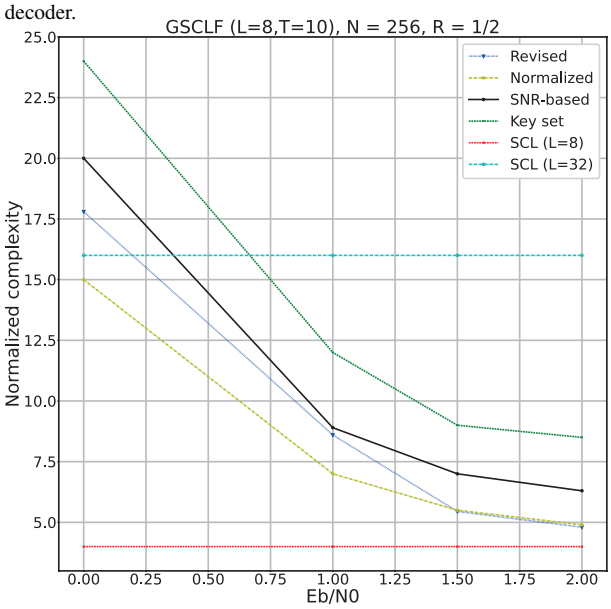

Fig. 8. Complexity of critical sets construction for GSCLF ($L = 8, T = 10$) decoder.

the oracle-based set has a high computational complexity, although it is more efficient than the heuristics-based methods. Despite the high complexity of the segmentation critical set, its performance is not impressive.

The belief propagation and subblocks critical sets have the highest complexity rates, and for small E_b/N_0 values, they can be several orders of magnitude more complex than the naive method. However, subblocks demonstrate excellent performance, which explains the observed complexity-performance tradeoff. On the other hand, the belief method is also promising but requires some parameter optimization.

It should be noted that SCF is a basic decoding algorithm that utilizes flipping strategies. Thus, its basic critical set has relatively low complexity, which is only three times higher than the basic SC decoder. However, the application of different critical set strategies may increase the complexity of SCF, and in some cases, it may approach the complexity of more sophisticated decoding methods (e.g., list-based methods). Nevertheless, the superior error correction performance of the subblock method make it a flexible algorithm whose complexity can be reduced by choosing optimal subtrees with suitable code rates.

SCLF, GSCLF: In Figs. 7 and 8, it can be observed that each critical set, when used with SCLF and GSCLF at low signal levels, outperforms even the relatively complex SCL-32 method. The SNR-based method exhibits the highest complexity indicators. Furthermore, the critical set normalization method was found to be in the lead when a large number of attempts were made to study performance characteristics. However, the list-based method was also comparable to it in terms of performance. With regards to computational complexity, it can be noted that the list-based algorithm reaches the limit of the SCL-32 method with 10 additional attempts. Therefore, despite the moderate number of decoding attempts and the increased complexity required to perform all operations, the list-based method provides the best value in terms of computational complexity.

It is recommended to use a list-based critical set for this decoding method, which is the least laborious and provides fairly good performance. To improve the performance and reliability of the decoder, it is recommended to use the normalized method with an accurate adjustment of the factor parameter η.

GSCF: Now, let us examine the DSCF decoder in Fig. 8. The segmentation set exhibits high performance, particularly for the dynamic method. This is attributed to the segmentation of critical indices during recalculation, resulting in refined flips that can enhance overall performance. It is noteworthy that the segmented set also demonstrates the best complexity results

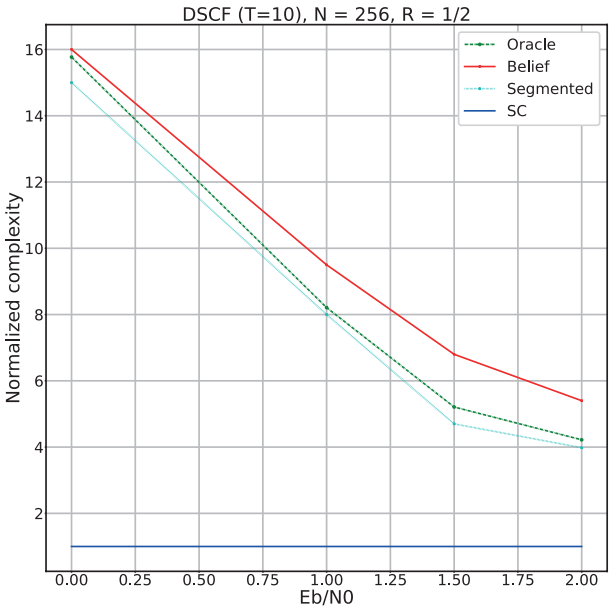

Fig. 9. Complexity of critical sets construction for DSCF ($T = 10$) decoder.

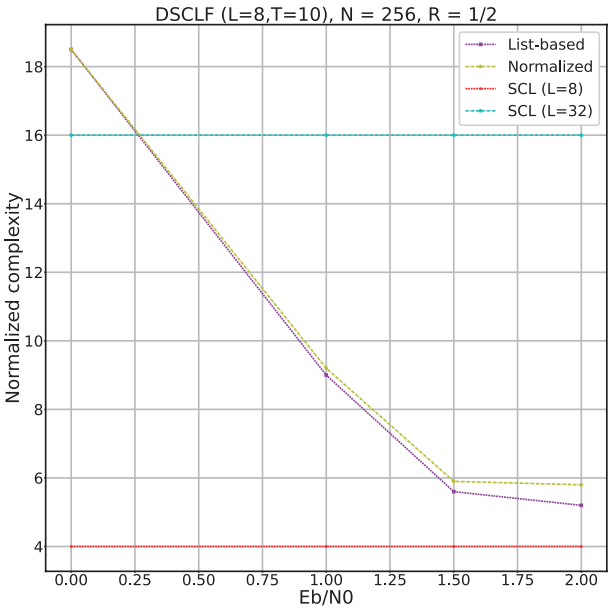

Fig. 10. Complexity of critical sets construction for DSCLF ($L = 8, T = 10$) decoder.

for $T = 10$.

DSCLF: Let's examine the performance of the DSCLF decoder in Fig. 10. The list-based version of the set leads to an improvement in complexity. However, in some cases, a simpler method may perform efficiently and have lower complexity. This is an interesting scenario, as it is only observed in this decoder and for a specific message length where a simplified version of constructing a critical set is more efficient than a refinement strategy.

As seen in the figure, the DSCLF algorithm achieves a complexity value equivalent to SCL-32 and even surpasses it. This indicates a high degree of decoder efficiency in terms of its complexity.

Although the performance of the list-based method for $T = 10$ is slightly lower than the normalized method, the complexity of both methods for constructing the critical set is comparable for all parameters.

Therefore, we can conclude that the use of either method is based on the same metric, but normalization offers a significant increase in performance relative to the list-based set, except for certain special cases where the opposite holds true. The list-based method is preferred for constructing the critical set in this work since it is a simpler option that can improve both labor intensity and performance in certain circumstances.

C. Performance Analysis of Decoders

Let us analyze how different decoders perform in terms of error rates and what is the optimal solution considering memory and time complexity. The error correction performance and normalized complexity for all described methods are visualized in Fig. 11.

The basic SC-decoder exhibits the worst error correction capability, and all other algorithms perform more efficiently as the SNR increases. The SCL decoder shows a significant improvement in correction ability compared to the SC decoder. For SCL-4 and SCL-8, we observe such improvement only when compared to the basic SC-decoder. However, when using the SCL-32 algorithm, a significant improvement in performance is visible, as larger sheets work more correctly, and CRC checks for $L = 32$ provide greater accuracy than using $L = 8$.

The performance of GSCLF approaches that of SCF at $T = 10$, which is an excellent result for this decoder but requires more computing power. The generalized method is quite accurate due to its ability to decode special nodes unambiguously before running the algorithm itself, as the theoretical values of the decoded sequence for the information node or SPC node have already been calculated. At $T = 10$, the GSCLF method shows particularly high-quality performance, providing FER of approximately 10^{-5}, which is the best result among all decoders. For a large number of attempts, SCLF is also a good method, which can equal GSCLF, but only at small values of E_b/N_0.

Thus, the most stable decoding method is GSCLF, and SCLF and DSCF methods also show good performance. Therefore, it is recommended to use the generalized list flipping method with a revised critical set oriented to handle some type of special nodes for optimal performance.

D. Complexity Analysis of Decoders

In Fig. 12, we will compare various decoding methods in terms of their complexity to determine the optimum solution. To facilitate comparison, the constant values of the SC and SCL-x decoders were normalized.

Among the considered decoders, the SCF method stands out as the fastest and least labor-intensive. This method even achieves lower complexity than SCL-8 while having higher performance than the basic list method as the SNR increases. With an increase in the number of additional attempts, this

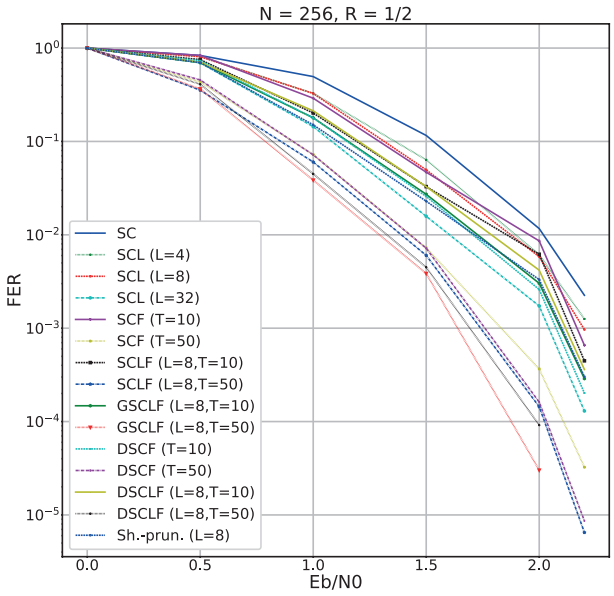

Fig. 11. Performance of different decoders with optimal critical sets.

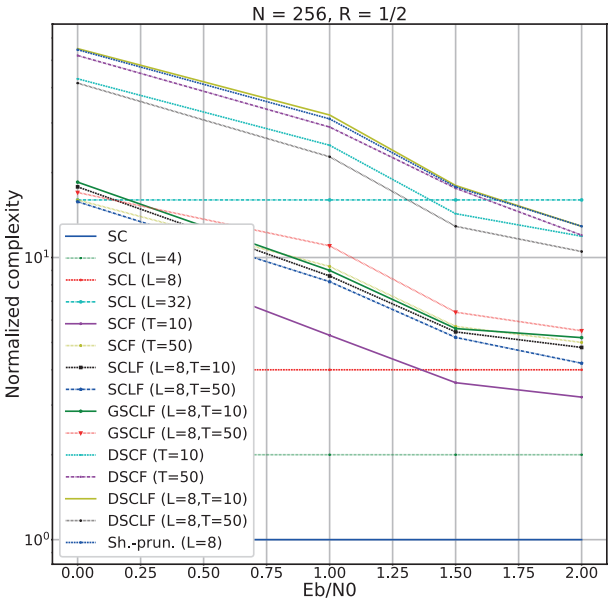

Fig. 12. Complexity of different decoders with optimal critical sets.

method is faster than list methods, and with a large SNR, SCF with 50 attempts is less complex than SCL-32. As seen in the performance plots, SCF reaches SCL-32 FER values at lower complexity than the list method. This is a general advantage of flipping algorithms.

The shifted-pruning method has a low enough complexity for a list method. For large SNRs, this method is comparable to SCL-8, but its complexity increases significantly. This approach turns out to be more inefficient than other methods with a similar value for the number of additional attempts.

The SCLF method has a rather high computational complexity since it lacks the optimizations inherent in dynamic and generalized methods. Interestingly, SCLF is roughly comparable in computational complexity to SCL-16. Since the performance of this method is not always optimal and often loses to more flexible list methods, it can be concluded that the method has a rather high overhead at an average level of performance.

For the GSCLF algorithm, we can see that its complexity is comparable to DSCF, while providing the highest level of performance. It should also be noted that it requires fewer resources due to the decoding of special generalized nodes, so its complexity is lower than SCLF.

In summary, considering both performance and computational complexity, the GSCLF and DSCLF decoding methods are optimal, providing lower complexity than both the SCLF method and the basic SCL decoders. Table 3 describes the average number of "cheap" operations from the C_1 class, "expensive" operations from the C_2 class, and their total amount for each decoder. Although all decoders exhibit similar behavior, they differ in the types of operations used. The SC method requires the minimum number of operations. The SCL and SCF methods, on the other hand, require a relatively large total number of operations, but they do not use expensive operations such as exponentials and logarithms, resulting in a smaller number of C_2 operations.

TABLE III
AVERAGE NUMBER OF OPERATIONS FOR DIFFERENT CODE LENGTH $N = \{64, 256, 1024\}$ AND CODE RATIO $R = 1/2$ WITH FER $= 10^{-2.5}$.

Algorithm	N	C_1	C_2	Total ops.
SC	64	13.30	3.89	17.19
	256	41.16	4.07	45.23
	1024	85.55	11.81	97.36
SCL ($L = 4$)	64	52.17	7.96	60.13
	256	173.17	14.09	187.26
	1024	503.34	57.06	560.40
SCL ($L = 8$)	64	109.73	12.58	122.31
	256	304.67	25.15	329.82
	1024	661.11	90.51	751.62
SCL ($L = 32$)	64	445.10	39.75	484.85
	256	812.54	97.55	910.09
	1024	3513.42	415.52	3928.94
SCF ($T = 10$)	64	92.31	14.90	107.21
	256	277.31	38.23	315.54
	1024	852.30	62.55	914.85
SCLF ($L = 8, T = 10$)	64	115.20	11.43	126.63
	256	322.56	48.13	370.69
	1024	951.10	135.42	1086.52
GSCLF ($L = 8, T = 10$)	64	98.99	3.72	102.71
	256	300.07	9.12	309.19
	1024	840.16	30.14	870.30
DSCF ($T = 20$)	64	114.58	31.92	146.50
	256	311.54	76.10	387.64
	1024	800.15	155.88	956.03
DSCLF ($L = 8, T = 20$)	64	131.20	20.86	152.06
	256	324.85	83.49	408.34
	1024	1041.33	197.10	1238.43
Sh.-prun. ($L = 8$)	64	139.67	36.55	176.22
	256	361.85	89.90	451.75
	1024	1220.65	204.44	1425.09

VII. RECOMMENDATIONS AND OBSERVATIONS

After conducting a thorough investigation of different critical sets and decoding methods, we have formulated several recommendations on the optimal usage of bit-flipping methods.

- The basic SC decoder is the most straightforward and fastest solution, but it is also the most unreliable method of decoding due to its lack of flipping or listing ability. Despite this, the SC method is applicable to many types of flipping methods and works well in combination with prospective critical sets to provide a higher error correction performance.

- SCL decoder is strongly dependent on its L parameter. With a trivial case ($L = 1$), this method is a fast, unreliable SC decoder. However, using CRC-checking frameworks and several candidates slightly increases its performance. The most optimal length of the list in this paper is $L = 8$, providing a good trade-off between complexity and performance. The SC decoder provides a lower bound for complexity and an upper bound for performance, while the SCL-32 provides an upper bound for complexity but the best performance.
- SCF method provides good complexity for all values of code ratio and code length. As R increases, it achieves the complexity of the SCL-8 decoder. We recommend using it for small code length N. While extra attempts affect the complexity and make it comparable with listing approaches, the correct choice of the critical set also makes a difference. Instead of a naive approach to choosing indices, we use subtrees that provide higher performance. Segmentation design is similar to the subblocks idea but has lower performance values.
- SCLF approach is the basic flipping solution for listing methods. We considered two different methods of critical set construction: Normalized with normalization factor $\nu = 0.7$ and a simplified version without the normalization process. It is interesting to note that the list-based critical set is comparable with normalized, and for a moderate number of attempts, it provides approximately the same performance as the complicated normalized version.
- The dynamical SCF decoder shows excellent performance. Also, flexible recalculation of the critical set provides lower complexity values. For instance, we recommend using a segmented critical set, which can easily be redefined by its design. The belief set has much lower performance and complexity values, so we do not recommend applying it for a dynamical approach, especially for a large number of attempts.
- For the DSCLF method, we considered only list-based and normalized critical sets as the most effective design, but it can be applied to different complex strategies with lower theoretical performance. This method is efficient and simple compared to other listed flipping approaches, but it is unreliable in the general case.
- GSCLF method is the best in terms of performance and complexity. Despite the rather complicated implementation and the use of generic nodes, it provides fast decoding with minimal memory and other resources. Thus, the proposed experimental method did not show its effectiveness in this study. To summarize, we can say that the GSCLF method is more reliable than DSCLF/DSCF approaches and based on special nodes' fast decoding, which provides higher performance with less complexity.
- We also used the subblocks critical set as the most effective design for the shifted-pruning method. However, we cannot say that this approach is the most effective and reliable because its performance is generally lower than that of other decoders. Additionally, this method provides the worst complexity values.

VIII. Conclusion

Polar encoding has emerged as a promising direction in the 5G communication system [37]. Therefore, the development of efficient and flexible decoding methods for polar codes is a significant problem in information theory. In this paper, we address this issue by considering various strategies and approaches for constructing decoding bases that provide optimal polar decoding. We evaluate not only the basic versions of each approach but also their improvements and novel ideas.

Our study focuses on the construction of critical sets, modifications, and restrictions for decoding algorithms, as well as the applicability of different critical sets to specific decoding strategies. Although we have made progress in identifying the best strategy for constructing critical sets in terms of complexity and performance trade-off, this remains a priority task that requires further investigation in future studies.

Furthermore, we need to explore the efficiency ratio for different path metrics and data structures. Among all the considered decoders, the implementations of GSCLF and DSCLF appear to be the most promising. We also need to investigate how combining different critical sets and acceptable decoding methods can achieve higher error correction performance. In this paper, we evaluated only several approaches and combinations of set evaluation and their applicability for different decoding strategies.

We examined the main basic methods for decoding polar codes, as well as flipping methods that optimize calculations and provide higher accuracy by using additional attempts. Our study shows that the generalization and dynamic approach are effective in reducing memory overhead and the number of elementary operations. We also considered various approaches to the construction of critical sets, which enable us to make a more correct choice of the bit or set of bits to be inverted.

Increasing the number of attempts T for flipping methods or the number of elements in the list method can increase the probability of finding the desired word and performing correct decoding. However, optimizing the construction of critical sets and finding the optimal decoder for a particular set are crucial recommendations, as even with relatively low memory costs or computational complexity, we can achieve the highest performance.

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, 2009.
[2] I. Tal and A. Vardy, "List decoding of polar codes," in *Proc. IEEE ISIT*, 2011.
[3] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.
[4] R. Wade and M. Jadhav, "Study of polar codes using BPSK," *Int. J. Eng. Research Technol.*, vol. 7, no. 7, 2018.
[5] S. A. Hashemi, N. Doan, T. Tonnellier, and W. J. Gross, "Deep-learning-aided successive-cancellation decoding of polar codes," in *Proc. Asil. Conf. on Sign., Sys., and Comp.*, 2019.
[6] W. W. Peterson and D. T Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, no. 1, pp. 228–235, 1961.
[7] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, 2012.
[8] O. Afisiadis, A. B.-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Proc. IEEE ACSSC*, 2014.

[9] Y. Yongrun, P. Zhiwen, L. Nan, and Y. Xiaohu, "Successive cancellation list bit-flip decoder for polar codes," in *Proc. IEEE WCSP*, 2018.

[10] F. Ivanov, V. Miroshnik, and E. Krouk, "Improved generalized successive cancellation list flip decoder of polar codes with fast decoding of special nodes," *J. Comm. and Net.*, vol. 23, no. 6, 2021.

[11] H. Vangala, E. Viterbo, and Y. Hong, "A comparative study of polar code constructions for the AWGN channel", *arXiv, abs/1501.02473*, 2015.

[12] P. Giard *et al.*, "Hardware decoders for polar codes: An overview," in *Proc. IEEE ISCAS*, 2016.

[13] F. Ercan, C. Condo, S. A. Hashemi, and W. J. Gross, "On error-correction performance and implementation of polar code list decoders for 5G," in *Proc. Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2017.

[14] A. Balatsoukas-Stimming, P. Giard, and A. Burg, "Comparison of polar decoders with existing low-density parity-check and turbo decoders," in *Proc. IEEE WCNCW*, 2017.

[15] P. Giard, "High-speed decoders for polar codes," Ph.D.dissertation, McGill University, Montreal, Canada, 2016.

[16] Y. Yuyu *et al.*, "Noise-aided belief propagation list bit-flip decoder for polar codes," in *Proc. IEEE WCSP*, 2020.

[17] F. Kotov, F. Ivanov, and Z. Alexey, "Method of critical set construction for successive cancellation list decoder of polar codes based on deep learning of neural networks," *SSRN Elec. J.*, 2022

[18] Z. Zhang, K. Qin, L. Zhang, H. Zhang, and G. T. Chen, "Progressive bit-flipping decoding of polar codes over layered critical sets," in *Proc. IEEE GLOBECOM*, 2017.

[19] Z. Zhang, K. Qin, L. Zhang, and G. T. Chen, "Progressive bit-flipping decoding of polar codes: A critical-set based tree search approach," *IEEE Access*, vol. 6, pp. 57738–57750, 2018.

[20] M. Rowshan and E. Viterbo, "Improved list decoding of polar codes by shifted-pruning," in *Proc. IEEE ITW*, 2019

[21] M. Rowshan and E. Viterbo, "Shifted pruning for path recovery in list decoding of polar codes," in *Proc. IEEE CCWC*, 2021.

[22] M. Hanif and M. Ardakani, "Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes," in *IEEE Commun. Lett.*, vol. 21, no. 11, pp. 2360–2363, 2017

[23] F. Ercan, T. Tonnellier, N. Doan, and W. J. Gross, "Practical dynamic SC-Flip polar decoders: Algorithm and implementation," *IEEE Trans. Signal Process.*, vol. 68, pp. 5441–5456, 2020.

[24] F. Ercan, T. Tonnellier, and W. J. Gross, "Energy-efficient hardware architectures for fast polar decoders," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 67, no. 1, pp. 322–335, 2020.

[25] X. Qiao, H. Cui, J. Lin, and Z. Wang, "Reducing search complexity of dynamic SC-Flip decoding for polar codes," in *Proc. ICCC*, 2021.

[26] M. Rowshan and E. Viterbo, "Efficient partial rewind of successive cancellation-based decoders for polar codes," *IEEE Trans. Commun.*, vol. 70, no. 11, pp. 7160–7168, 2022.

[27] F. Ercan, C. Condo, and W. J. Gross, "Improved bit-flipping algorithm for successive cancellation decoding of polar codes," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 61–72, 2019.

[28] L. Chandesris, V. Savin, and D. Declercq, "Dynamic-SCFlip decoding of polar codes," *IEEE Trans. Commun.*, vol. 66, no. 6, pp. 2333–2345, 2018.

[29] Y. Shen, A. Balatsoukas-Stimming, X. You, and C. Zhang, "Dynamic SCL decoder with path-flipping for 5G polar codes," *IEEE Wireless Comm. Lett.*, vol. 11, no. 2, pp. 391–395, 2022.

[30] Y.-H. Pan, C.-H. Wang, and Y. -L. Ueng, "Generalized SCL-Flip decoding of polar codes," in *Proc. GLOBECOM*, 2020

[31] C. Condo, V. Bioglio, and I. Land, "Generalized fast decoding of polar codes," in *Proc. IEEE GLOBECOM*, 2018.

[32] R. Gallager, "Low-density parity-check codes," *IRE Trans. on Inf. Theory*, vol. 8, no. 1, 1962.

[33] R. W. Hamming, "Error detecting and error correcting codes," *Bell Sys. Tech. J.*, vol. 29, no. 2, pp. 147–160, 1950

[34] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 946–957, 2014.

[35] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive cancellation list decoders for polar codes" *IEEE Trans. Signal Process.*, vol. 65, no. 21, pp. 5756–5769, 2017.

[36] M. Hanif, M. H. Ardakani, and M. Ardakani, "Fast list decoding of polar codes: Decoders for additional nodes" in *Proc. IEEE WCNCW*, 2018.

[37] V. Bioglio, "Design of polar codes in 5G new radio," *IEEE Commun. Surveys Tuts*, vol. 23, no. 1, pp. 29–40, 2020.

[38] C.-H. Chen, C.-F. Teng, and A.-Y. A. Wu, "Low-complexity LSTM assisted bit-flipping algorithm for successive cancellation list polar decoder," in *Proc. IEEE ICASSP*, 2020.

[39] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Process.*, vol. 61, no. 2, pp. 289–299, 2013.

[40] Y. Fan and C. Y. Tsui, "An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation," *IEEE Trans. Signal Process.*, vol. 62, no. 12, pp. 3165–3179, 2014.

[41] F. Ercan, C. Condo, S. A. Hashemi, and W. J. Gross, "Partitioned successive-cancellation flip decoding of polar codes," in *Proc. IEEE ICC*, 2018.

[42] C.-F. Teng and A.-Y. A. Wu, "Convolutional neural network-aided tree-based bit-flipping framework for polar decoder using imitation learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 300–313, 2020.

[43] Y. Peng, X. Liu, and J. Bao, "An improved segmented flipped successive cancellation list decoder for polar codes," in *Proc. IEEE ICC*, 2020.

[44] U. Lee, J. H. Roh, C. Hwangbo, and M. H. Sunwoo, "A uniformly segmented SC-Flip decoder for polar codes with memory reduction methods," in *Proc. IEEE ISCAS*, 2021.

[45] F. Ercan and W. J. Gross, "Fast thresholded SC-Flip decoding of polar codes" in *Proc. IEEE GLOBECOM*, 2020.

[46] 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. WCNC*, 2019.

[47] S. Li, Y. Deng, X. Gao, H. Li, L. Guo, and Z. Dong, "Generalized segmented bit-flipping scheme for successive cancellation decoding of polar codes with cyclic redundancy check," *IEEE Access*, vol. 7, pp. 83424–83436, 2019.

[48] J. Bao, S. Lin, and X. Liu, "An improved successive cancellation list flip decoder for polar codes based on key sets," in *Proc. IEEE ISMICT*, 2021.

[49] X. Liu, S. Wu, Y. Wang, N. Zhang, J. Jiao, and Q. Zhang, "Exploiting error-correction-CRC for polar SCL decoding: A deep learning-based approach," *IEEE Trans Cog. Commun. Netw.*, vol. 6, no. 2, pp. 817–828, 2019.

[50] V. Miloslavskaya, B. Vucetic, Y. Li, G. Park, and O. -S. Park, "Recursive design of precoded polar codes for SCL decoding," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 7945–7959, Dec. 2021.

[51] V. Miloslavskaya and B. Vucetic, "Design of short polar codes for SCL decoding," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 6657–6668, 2020.

[52] I. Timokhin and F. Ivanov, "On the improvements of successive cancellation Creeper decoding for polar codes", *Digital Signal Process.*, vol. 137, 2023.

[53] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast simplified successive-cancellation list decoding of polar codes," in *Proc. IEEE WCNCW*, 2017.

[54] G. Trofimiuk, N. Iakuba, S. Rets, K. Ivanov, and P. Trifonov, "Fast block sequential decoding of polar codes," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 10988–10999, 2020.

[55] N. Doan, S.-A. Hashemi, and W.-J. Gross "Fast successive-cancellation list flip decoding of polar codes" *IEEE Access*, vol.10, pp. 5568–5584, 2020.

[56] V. Miloslavskaya and P. Trifonov, "Sequential Decoding of Polar Codes," in *IEEE Commun. Lett.*, vol. 18, no. 7, pp. 1127–1130, 2014.

[57] S.-A. Hashemi *et al.*, "Successive syndrome-check decoding of polar codes" in *Proc. ACSSC*, 2021.

[58] S.-A. Hashemi, C Condo, M.Mondelli, and W.-J. Gross, "Rate-flexible fast polar decoders" *IEEE Trans. Signal Process.,* vol 67, no 22, pp. 5689–5701, 2019.

[59] C. Condo, V. Bioglio, and I. Land, "SC-Flip decoding of polar codes with high order error correction based on error dependency," in *Proc. IEEE ITW*, 2019.

[60] G. Coppolino, C. Condo, G. Masera, and W. J. Gross, "Efficient operation scheduling in successive-cancellation-based polar decoders," in *Proc. SiPS*, 2018.

[61] S. A. Hashemi, C. Condo, and W. J. Gross, "List sphere decoding of polar codes," *Proc. ACSSC*, 2015.

[62] H. Zhou, W. J. Gross, Z. Zhang, X. You, and C. Zhang, "Efficient sphere polar decoding via synchronous determination," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 6777–6781, 2020.

[63] C. Pillet, C. Condo, and V. Bioglio, "Fast-SCAN decoding of polar codes" in *Proc. IEEE ISTC*, 2021.

[64] X. Hu, H. Hou, X. Jiang, S. Sun, G. LIANG, and S. Han, "An optimized successive cancellation list decoder for polar codes combined with critical set," in *Proc. IEEE IWCMC*, 2022.

[65] D. Yang and K. Yang, "Error-Aware SCFlip decoding of polar codes" *IEEE Access*, vol. 8, pp. 163758–163768, 2020.

I. Timokhin at the moment is a postgraduate student of the Higher School of Economics in the direction of Information Security, and also provides researches in the laboratory of Cyber-Physical Systems and the Internet of Things. He also works at Huawei's Moscow Research Center, analyzing big data, complex query execution efficiency, and computing resource optimization. Professional interests are artificial intelligence, hash function research, polar coding, and graph theory.

F. Ivanov is currently an Associate Professor in the Department of Electronic Engineering at National Research University Higher School of Economics, Moscow, Russia. He received his M.S. degree in Mathematics in 2011 from Far Eastern Federal University, Vladivostok, Russia and the and Ph.D. degree in Computer Engineering and Theoretical Informatics from Moscow Institute of Physics and Technologies (Moscow, Russia) in 2014. He also works as a Fellow Researcher at Institute for Information Transmission Problems, Russian Academy of Science from 2011. His research interest includes communication theory, polar codes, LDPC codes, concatenated codes, convolutional codes, and non-orthogonal multiple access.