SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

July 2026, Vol. 69, Iss. 7, 172301:1–172301:15
https://doi.org/10.1007/s11432-025-4725-3

Tanner-graph-assisted successive cancellation decoding for large kernel polar codes

Yangyang LIU, Yu ZHANG, Liuchang YANG, Lixia XIAO & Tao JIANG*

Research Center of 6G Mobile Communications, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China

Received 17 July 2025/Revised 27 October 2025/Accepted 16 December 2025/Published online 31 March 2026

Abstract Large kernel polar codes demonstrate exceptional error-correction capabilities in finite-length coding, making them promising candidates for enabling 6G's ultra-high reliability and ultra-low latency communication (URLLC) requirements. However, the decoding complexity exhibits exponential growth relative to the kernel dimension. In this paper, we study Tanner-graph-assisted (TGA) serial decoding to further reduce complexity without sacrificing error-correction performance. We first establish the standard for the Tanner graph to satisfy serial decoding and give rigorous proof. In particular, a gamified worm search scheme is designed, which makes the proposed standard practical for screening Tanner graphs by accurately tracking the worm's running trajectory in the node matrix. Then, we construct a low-complexity serial TGA successive cancellation (TGA-SC) decoder. The decoder takes the kernel matrix that satisfies both the standard and the optimal polarization exponent as the Tanner graph. Moreover, the serial decoding equations for arbitrary dimensional linear binary kernels are derived. Numerical results demonstrate that our scheme achieves significant error-correction improvement compared to classic polar codes while maintaining similar complexity.

Keywords polar codes, large kernels, successive cancellation decoding, Tanner graph, URLLC

Citation Liu Y Y, Zhang Y, Yang L C, et al. Tanner-graph-assisted successive cancellation decoding for large kernel polar codes. Sci China Inf Sci, 2026, 69(7): 172301, https://doi.org/10.1007/s11432-025-4725-3

1 Introduction

Polar codes, as theoretically optimal codes capable of reaching Shannon's capacity limit, have been successfully integrated into 5G New Radio standards for reliable channel control [1, 2]. The foundational structure of polar codes stems from the polarization effect generated through Kronecker product operations on two-dimensional kernel matrices. However, when applied to 6G's ultra-reliable low-latency communication (URLLC) requirements [3, 4], existing 5G polar code implementations face significant challenges. This limitation arises from 6G URLLC's need for shorter code lengths to reduce transmission delays, which substantially diminishes channel polarization effects and consequently degrades polar code performance [5]. This critical situation has created an urgent demand for enhancing the polarization efficiency of polar codes under short blocklength conditions.

Large kernel polar codes, designed to enhance polarization through the construction of high-dimensional polarization kernel matrices [6], have gained growing research interest. Ref. [7] demonstrated that increasing kernel size improves polarization performance, while also establishing necessary and sufficient conditions for such matrices to achieve polarization. Complementary approaches include the use of decomposition techniques to optimize binary kernels and the development of non-binary kernels with higher polarization exponents derived from Reed-Solomon codes. Further exploration in [8], extended to linear and nonlinear binary kernels, achieves record polarization exponents for dimensions up to 16. Collectively, these advancements have strengthened polar codes' asymptotic error-correction capabilities, positioning them as increasingly competitive against state-of-the-art channel coding methods.

Despite the promising theoretical results, large kernels pose challenges for implementing the polar decoders. Generally, the complexity of a straightforward successive cancellation (SC) decoder for length N polar codes based on kernel $\mathbf{F}_p$ scales as $O(2^p N \log_p N)$ [7], which increases exponentially with the kernel size p. Several approaches have been explored to mitigate the high decoding complexity of large kernel polar codes. Ref. [9] introduced an l-formula technique to simplify LLR expressions, though only formulations for $\mathbf{F}_3$ and $\mathbf{F}_4$ kernels were provided. In [10], an

* Corresponding author (email: tao.jiang@ieee.org)

approximate kernel processing method leveraging window decoding was proposed, accompanied by implementations for 16 and 32 dimensional kernels. These designs demonstrated reduced decoding complexity relative to $\mathbf{F}_2$-based polar codes at comparable error correction performance. Notably, not all kernels support this approach in an efficient way. Addressing this limitation, Ref. [11] developed a unified recursive trellis-based framework for kernel processing, employing the max operator to approximate the summation operator in the computation of kernel processing. However, both window decoding and recursive trellis necessitate a compromise between complexity and decoding accuracy. Recently, Ref. [12] presented a W-expression approach utilizing bit-channel transition probabilities, reducing SC decoder complexity to $O(p^2N\log_p N)$ while maintaining the error-correction benefits of large kernels. Despite this advancement, the method currently supports kernels up to dimension 16 and retains higher computational overhead than the classic polar code.

As an auxiliary tool, the Tanner graph can effectively reduce decoding complexity [13]. However, due to the requirement of simultaneously satisfying the kernel matrix and bit-by-bit serial decoding characteristics, existing solutions such as classical polar and the Tanner graph in [13] cannot be applied to SC decoding for large kernel polar codes. To our knowledge, there currently exists no research on Tanner graph designs tailored for SC decoding of large kernel polar, which remains an open problem.

In this paper, we explore Tanner-graph-assisted SC decoding for large kernel polar codes. The Tanner graph employed in large kernel SC encounters two primary challenges. First, the dual constraints imposed by the kernel matrix and bit-by-bit serial decoding necessitate rigorous deterministic standards to ensure Tanner graph configurations align with serial decoding principles. Second, given the kernel size, the potential candidate space of Tanner graphs may be huge, and it is very tricky to quickly search for one that satisfies the deterministic standards. We address these obstacles by investigating the Tanner graph construction and serial decoding framework design. The main contributions can be summarized as follows.

(1) We develop a theoretical deterministic standard for kernel Tanner graphs to concurrently satisfy constraints stemming from the polarization effect of the high-dimensional kernel matrix and the principles of bit-by-bit serial decoding. Furthermore, the proposed standard is formally validated through a tree structure analysis. To the best of our knowledge, this work represents the first systematic integration of Tanner graph theory to enhance serial decoding for large kernel polar codes.

(2) We introduce a gamified worm traversal algorithm designed to facilitate the practical application of the standard by meticulously monitoring the worm's movement path across node matrices. The algorithm is subsequently integrated into an automated kernel matrix constructor, which generates compliant polarization kernel matrices that adhere to serial decoding requirements. Leveraging this framework, we propose Tanner-graph-assisted successive cancellation list (TGA-SCL) decoding for large kernel polar and derive serial decoding equations applicable to linear binary kernels of arbitrary dimensions.

(3) Extensive numerical results validate that our proposed large kernel TGA-SCL decoder is capable of providing a significant performance gain over PCC polar codes standardized in 5G. Particularly, under an additive white Gaussian noise (AWGN) channel, when the kernel dimension $p = 15$ and the code length $N = 120$, a FER performance gain of 0.2 dB is obtained. In addition, the computational complexity of TGA-SCL decoding is $O(L\frac{2EN}{p}\log_p N)$, which is similar to that of classical polar codes.

The remainder contents are organized as follows. In Section 2, we introduce the basics of large kernel polar. In Section 3, we give and prove the standard on the kernel and a worm search strategy is proposed to meet the standard. In addition, the decoding equations for arbitrary dimensional kernel are derived. In Section 4, we elaborate the enhanced TGA-SCL decoding. Section 5 presents extensive simulation results. Finally, we conclude our paper in Section 6.

2 Preliminaries

2.1 Large kernel polar encoding

Large kernel polar codes can provide excellent polarization rate by constructing a kernel matrix with large error exponent (EE). Specifically, the error exponent of a kernel $\mathbf{F}_p$ is given by

$$EE(\mathbf{F}_p) = \frac{1}{p}\sum_{i=0}^{p-1}\log_p(D_i), \tag{1}$$

where D_i $(i = 0, \cdots, p-1)$ is the partial distance, and p is the size of the kernel matrix.

Similar to polar encoding, the codeword x of large kernel polar codes can also be obtained from the generator matrix $\mathbf{G}_N$, which is expressed as $x_1^N = u_1^N \mathbf{G}_N = u_1^N \mathbf{F}_l^{\otimes n}$, where code length $N = l^n$, and $\mathbf{F}_l$ denotes polarization kernel with a size of $l \times l$. The code structure of large kernel polar codes fulfills an equivalent role to the generator matrix, embodying the fundamental correlation between input vectors and codewords. As demonstrated in Figure 1, using a kernel case of size 5, the structure organizes P stages arranged right-to-left, with each stage comprising N/l identically sized kernels. The critical challenge in designing such a structure lies in determining the interconnection vectors between consecutive stages, which is summarized as follows.

(1) Calculate the connection vectors between stage 1 and stage $P-1$, which can be calculated as

$$w_i = (\alpha_i, \alpha_i + \beta_{i+1}, \alpha_i + 2 \times \beta_{i+1}, ..., \alpha_i + (N/\beta_{i+1} - 1)\beta_{i+1}), \tag{2}$$

where, $i \in [2, P-1]$, $\beta_1 = 1$, $\beta_i = \prod_{j=1}^{i-1} lj$, $\alpha_i = (1, li+1, 2 \times li+1, ..., (\beta_i - 1) \times li + 1, ..., li, li + li, 2 \times li + li, ..., (\beta_i - 1) \times li + li)$.

(2) Compute the connection vector w_P between the $(P-1)$-th stage and the P-th stage, which is defined as

$$w_P = (1, lP+1, 2 \times lP+1, ..., (\beta_P - 1) \times lP + 1, ..., lP, lP + lP, 2 \times lP + lP, ..., (\beta_P - 1) \times lP + lP). \tag{3}$$

(3) Derive the connection vector w_1 between codeword x and the initial stage. This vector ensures precise alignment between codewords generated via the generator matrix and those derived from the Tanner graph representation.

2.2 Large kernel polar decoding

With a code structure, SC decoding can be performed on it. However, the absence of specialized analysis tools, such as the Tanner graph of polar codes, leaves kernel operations in the code structure as black boxes processes [13]. Consequently, SC decoding for large kernel polar depends critically on the fundamental formula derived from the generic kernel matrix $\mathbf{F}_l$,

$$W_l^{(i)}(y_1^l, u_1^{i-1}|u_i) = \frac{1}{2^{l-1}} \sum_{u_{i+1}^l} W_l(y_1^l|u_1^l) = \frac{1}{2^{l-1}} \sum_{u_{i+1}^l} W(y_1|(u_1^l F_l)_1) \cdots W(y_l|(u_1^l F_l)_l), \tag{4}$$

where $u_{i+1}^l \in \{0,1\}^{l-i}$. The LLR of the i-th bit channel is defined as

$$LLR_l^{(i)} = \ln \frac{W_l^{(i)}(y_1^l, u_1^{i-1}|u_i = 0)}{W_l^{(i)}(y_1^l, u_1^{i-1}|u_i = 1)} = \ln \frac{\sum_{u_{i+1}^l} W(y_1|(u_{1,u_i=0}^l F_l)_1) \cdots W(y_l|(u_{1,u_i=0}^l F_l)_l)}{\sum_{u_{i+1}^l} W(y_1|(u_{1,u_i=1}^l F_l)_1) \cdots W(y_l|(u_{1,u_i=1}^l F_l)_l)}, \tag{5}$$

where $u_{1,u_i=1}^l$ represents $(u_1^{i-1}, u_i = 1, u_{i+1}^l \in \{0,1\}^{l-i})$. As the kernel size increases, the complexity of (5) will increase exponentially. Clearly, directly implementing SC decoding is infeasible. Numerous studies have aimed to simplify the computational demands of (5), including approaches like l-formula and W-expressions [12], yet these solutions remain limited to kernels of size 16 or smaller. Compounding the issue, even these optimized methods maintain substantially higher computational requirements compared to classic polar code SC decoding.

From an alternative viewpoint, Ref. [13] illuminated the structural intricacies of the code structure by constructing a Tanner graph aligned with the kernel matrix, subsequently introducing Tanner-graph-assisted belief propagation list (TGA-BPL) decoding for large kernel polar. The TGA-BPL decoding process initiates at the rightmost nodes of the Tanner graph, iteratively computing leftward information for key nodes and input-side nodes across each layer of the kernel matrix. Upon reaching the leftmost nodes, the algorithm reverses direction to calculate rightward information for relevant nodes layer-by-layer. The decoding terminates upon reaching the maximum iteration count, yielding the estimated codeword sequence $\hat{u}_1^N$. Figure 2 exemplifies a Tanner graph for code length $N = 5$. Here, nodes a and b are initialized with zero-valued left and right information, and serve as key nodes in the decoding process. All key nodes are gathered into a set $\mathcal{Q}$, and needs to satisfy the following constraints: (i) the L and R information of any Q_i can be expressed by Q_j $(j \neq i)$, as well as the input and output nodes of the kernel Tanner graph; (ii) the number of KN should be as small as possible. The left information L_i $(i = 6, 7, \cdots, 10)$ and the right information R_j $(j = 1, 2, \cdots, 5)$ are initialized as

$$L_i = LLR(y_i), \tag{6}$$

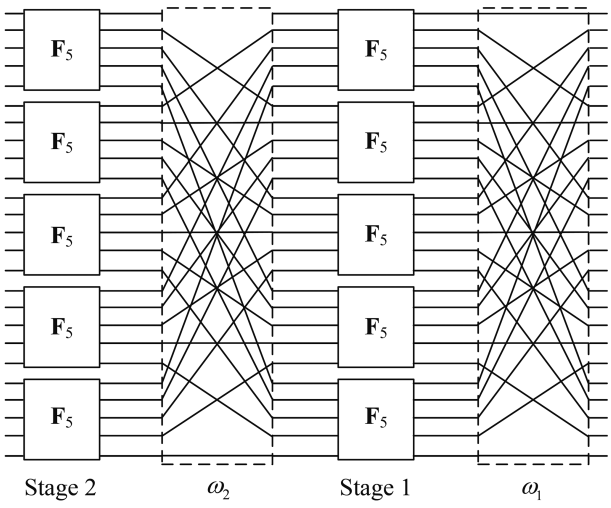

Figure 1 Code structure of the $\mathbf{F_5}$-based polar code with $N = 25$.

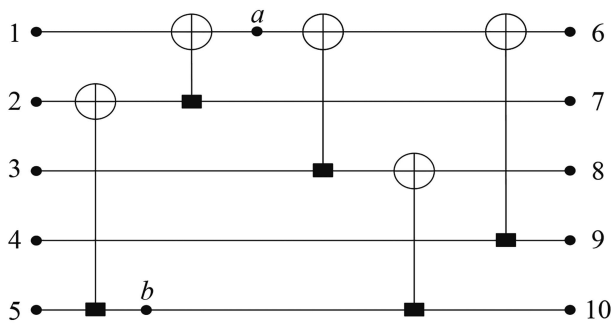

Figure 2 Kernel Tanner graph corresponding to kernel size 5.

$$R_j = \begin{cases} 0, \text{ if } i \in \mathcal{I}, \\ +\infty, \text{ if } i \in \mathcal{I}^{\mathcal{C}}, \end{cases} \tag{7}$$

where $LLR(y_i)$ is the channel LLR received by the i-th node and $\mathcal{I}$ is the information bit index set, $\mathcal{I}^{\mathcal{C}} = \{i | i \in [N], i \notin \mathcal{I}\}$. The left information of the key nodes is calculated as

$$L_a = g(g(L_6, R_4 + L_9), R_3 + g(L_8, R_b + L_{10})), \tag{8}$$

$$L_b = L_{10} + g(R_3 + g(R_a, g(L_6, R_4 + L_9)), L_8), \tag{9}$$

where the function $g(x, y)$ is used to obtain the LLR of the lower left node in an XOR structure, and can be calculated as $g(x, y) = \ln \frac{1+e^{x+y}}{e^x+e^y}$. Furthermore, after possessing the L of nodes a and b, the L information of the input side nodes of the kernel Tanner graph can be derived:

$$L_1 = g(L_a, g(R_2, R_5 + L_b) + L_7), \tag{10}$$

$$L_2 = g(L_7 + g(R_1, L_a), R_5 + L_b), \tag{11}$$

$$L_3 = g(L_8, R_b + L_{10}) + g(R_a, g(L_6, R_4 + L_9)), \tag{12}$$

$$L_4 = L_9 + g(g(R_a, R_3 + g(L_8, R_b + L_{10})), L_6), \tag{13}$$

$$L_5 = L_b + g(R_2, L_7 + g(R_1, L_a)). \tag{14}$$

After updating the L information of the leftmost nodes of the kernel Tanner graph, the R information of the key nodes and the rightmost nodes is calculated from left to right in a similar process. The L and R information update process is continuously cycled until the maximum number of iterations is reached. The sum of the L and R information of the leftmost nodes is obtained as $LLR_i = L_i + R_i$, and the estimated decoding result is defined as

$$\hat{u}_i = \begin{cases} 0, & i \notin \mathcal{I}, \\ 1, & LLR_i < 0, i \in \mathcal{I}, \\ 0, & LLR_i \geqslant 0, i \in \mathcal{I}. \end{cases} \tag{15}$$

3 TGA-SC decoding for large kernel polar

It is a fact that both $\mathbf{F}_2$-based polar and TGA-BPL [13] inherently share a structural foundation in utilizing the Tanner graph, enabling their capacity for low-complexity serial or parallel decoding implementations. However, extending these Tanner graph methodologies to larger kernel serial SC decoding is not feasible. The Tanner-graph-assisted SC decoding for larger kernel serial decoding encounters two primary challenges. First, the dual constraints imposed by the kernel matrix and bit-by-bit serial decoding necessitate rigorous deterministic standards to ensure

Tanner graph configurations align with serial decoding principles. Second, the explosion of potential Tanner graph candidates for a given kernel size creates significant difficulty in quickly identifying one specifically designed for larger kernels. This section systematically addresses these two challenges and introduces a novel low-complexity Tanner-graph-assisted SC serial decoding framework specifically designed for larger kernels.

3.1 Standards of valid Tanner graphs

The Tanner graph can be regarded as an effective tool to assist decoding. Different channel coding schemes have different Tanner graphs. For example, the Tanner graph of an LDPC code is designed based on the check matrix and consists of check nodes and variable nodes [14]. The classical polar code is composed of the XOR structure corresponding to the kernel matrix of size 2 [15]. Although the Tanner graph is designed for a large kernel polar code in [13], it is only for parallel decoding. When it comes to SC decoding, due to the bit-by-bit serial decoding characteristics, the Tanner graph used for SC decoding has more stringent restrictions. Specifically, any decoded bit in the Tanner graph needs to be able to determine the current bit based on the estimated decoded bits [16]. For ease of understanding, taking the kernel Tanner graph in Figure 2 as an example, although it can be used for parallel BP, it cannot be directly employed for SC decoding. We assume that given the estimated bit $\hat{u}_1$ and the received channel LLR, one cannot use this to decode any of the nodes 2–5.

Given a polarization kernel matrix $\mathbf{F}_{p\times p}$ of size p, the corresponding kernel Tanner graph candidate space can be obtained through [13]. It is a challenge to determine whether a kernel Tanner graph can be SC decoded from the huge candidate space. Next, we propose the standard for determination.

Proposition 1. If the kernel Tanner graph satisfies SC decoding, then for any rightmost node $x_i \in \{x_1, x_2, \cdots, x_p\}$ of the kernel Tanner graph, the following conditions must be met:

$$\tau(\mathbf{I}_{u_j}, x_i) \leqslant 1. \tag{16}$$

Therein u_j $(j = 1, 2, \cdots, p)$ is the j-th leftmost node, $\mathbf{I}_{u_j} \in \{LLR_{x_1^N}, R_{u_1^N}\}$, and function τ is defined as

$$\tau(\mathbf{I}_{u_j}, x_i) = \begin{cases} 0, & x_i \cap \mathbf{I}_{u_j} = \emptyset, \\ \sum\limits_{q=1}^{|\mathbf{I}_{u_j}|} \varphi(\mathbf{I}_{u_j}(q), x_i), & \text{otherwise}, \end{cases} \tag{17}$$

where the function φ is computed as

$$\varphi(a, b) = \begin{cases} 1, & a = b, \\ 0, & \text{otherwise}. \end{cases} \tag{18}$$

Proof. We first analyze the decoding process of SC. An important feature is to estimate the LLR of each bit to be decoded serially, and obtain the decoding result by hard decision of the LLR. The LLR estimate of the current decoded bit i is related to two variables: (i) the LLR sequence $\boldsymbol{L}_i$ received by the channel; (ii) the bit sequence $\hat{\boldsymbol{u}}_i$. Assume that the estimated value of the i-th decoded bit is

$$LLR_i = h_p^{(i)}(\hat{\boldsymbol{u}}_i, \boldsymbol{L}_i), \tag{19}$$

where $\hat{\boldsymbol{u}}_i \in \hat{\boldsymbol{u}}_1^{i-1}$, $\boldsymbol{L}_i \in \boldsymbol{LLR}_1^N$. The functional description of the $h_p^{(i)}$ is that given the sequence $\hat{\boldsymbol{u}}_i$ and $\boldsymbol{L}_i$, SC decoding is performed to obtain the current decoded bit. From another perspective, Eq. (19) can be equivalently converted into decoding based on the Tanner graph and is described in detail in Subsection 3.3. Specifically, according to the LLR values $\boldsymbol{L}_i$ of some nodes on the rightmost side of the Tanner graph and the estimated decoding result $\hat{\boldsymbol{u}}_i$ on the leftmost side, the left information L, which represents the LLR, and the right information R, which implies the decoded bit, are transmitted through the Tanner graph structure to estimate the LLR of a node on the left side of the Tanner graph.

It can be found that the structure of the Tanner graph is the key factor affecting whether large kernel polar codes can perform SC decoding. In particular, the Tanner graph of a large kernel polar code is composed of several XORs, where the left information LLR transfer is defined as the tree structures (a) and (b) in Figure 3, and the right information partialsum transfer is shown as the tree structures (c) and (d) in Figure 3. The tree structure represents the root node's estimated dependency on its child nodes. For example, in an XOR structure, the upper left, lower left, upper right, and lower right nodes are denoted as 1, 2, 3, and 4, respectively. The left information

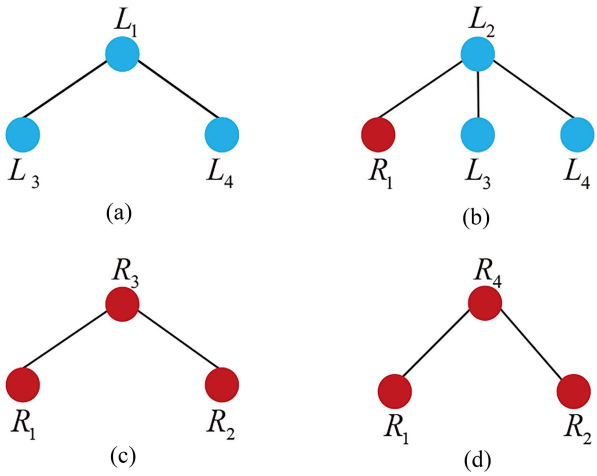

Figure 3 (Color online) Tree structure corresponding to a single XOR. (a) Upper left node; (b) lower left node; (c) upper right node; (d) lower right node.

L_2 calculation of node 2 depends on the partialsum R_1 of node 1, as well as the L_3 and L_4 in Figure 3(b), which can be described as

$$L_2 = \phi(R_1, L_3, L_4), \tag{20}$$

where the function ϕ is defined as

$$\phi(x, y, z) = (-1)^x \cdot y + z. \tag{21}$$

Given any leftmost node u_j $(j = 1, 2, \cdots, p)$ in the Tanner graph, take it as the root node, which is recorded as the first level of the tree, and continuously expand its children downward. For all nodes t at level l $(l = 1, 2, \cdots, T)$ in the tree, where T is the total number of levels, $|t| = 2^l$. Let the subset $I_{u_j}^{(l)}$ of t be

$$I_{u_j}^{(l)} = \{h | h \in R_{u_1^p} \text{ or } h \in L_{x_1^p}\}. \tag{22}$$

Note that for all nodes in $I_{u_j}^{(l)}$, they have no children. Therefore, the LLR estimate of the root node u_j will be uniquely determined by the information carried by set $\mathbf{I}_j$. In particular, $\mathbf{I}_j$ is expressed as

$$\mathbf{I}_{u_j} = \{I_{u_j}^{(1)}, I_{u_j}^{(2)}, \cdots, I_{u_j}^{(T)}\}. \tag{23}$$

According to (20), we can find that only after obtaining the left information L_1, can someone calculate the left information L_2. Therefore, L_1 and L_2 cannot exist at the same time, which is equivalent to the number of times L_3 or L_4 cannot appear more than twice simultaneously. Furthermore, if the current root node u_j satisfies (19), the number of left information of any node x_i $(i = 1, 2, \cdots, p)$ on the rightmost side of the Tanner graph that appears in $\mathbf{I}_{u_j}$ cannot exceed 1.

For ease of understanding, we give an example when the kernel dimension is 7, and its Tanner graph is shown in Figure 4. Take the leftmost node u_1 of the Tanner graph as the root. Since the calculation of the left information of u_1 depends on L_a and L_h, the root node u_1 has two descendants, L_a and L_h. Furthermore, L_a is calculated based on L_c and L_d, and L_c depends on the L_{x_1} and L_{x_2}, which are the LLR received by the channel. At this point, since nodes x_1 and x_2 are the rightmost nodes of the Tanner graph, the subsequent generations will not be expanded. The same process will eventually result in a tree with u_1 as the root node, as shown in Figure 5(a), where the set $\mathbf{I}_{u_1}$ is

$$\mathbf{I}_{u_1} = \{L_{x_1}, L_{x_2}, \cdots, L_{x_7}\}. \tag{24}$$

In addition, L_{x_i} is uniquely determined by the node x_i. Therefore, Eq. (24) can be simplified to $\mathbf{I}_{u_1} = \{x_1, x_2, \cdots, x_7\}$, which satisfies (16). So far we can conclude that node u_1 can perform SC decoding through (19).

Different from u_1, node u_3 also depends on the estimated decoding result, which is the right information R_{u_1} of u_1. Therefore, the root node u_3 has three descendants, namely R_{u_1}, L_a and L_h. The descendants of L_a and L_h are derived from the tree structure formed by the root node u_1. R_{u_1} has no descendants because it belongs to the leftmost node information of the Tanner graph. The tree with u_3 as the root node is shown in Figure 5(b), where the set $\mathbf{I}_{u_3}$ is

$$\mathbf{I}_{u_3} = \{R_{u_1}, L_{x_1}, L_{x_2}, \cdots, L_{x_7}\}. \tag{25}$$

Therefore, node u_3 also satisfies the SC decoding in (19).

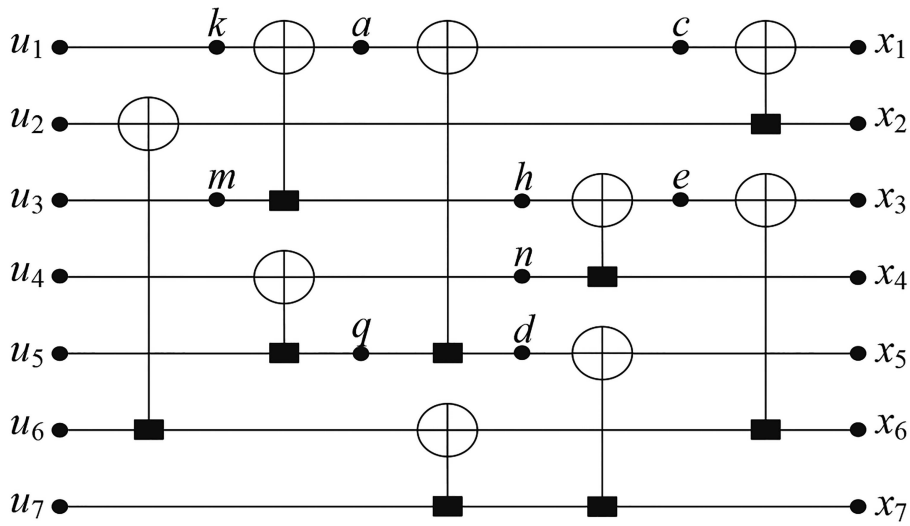

Figure 4 Kernel Tanner graph corresponding to $\mathbf{F}_7$.

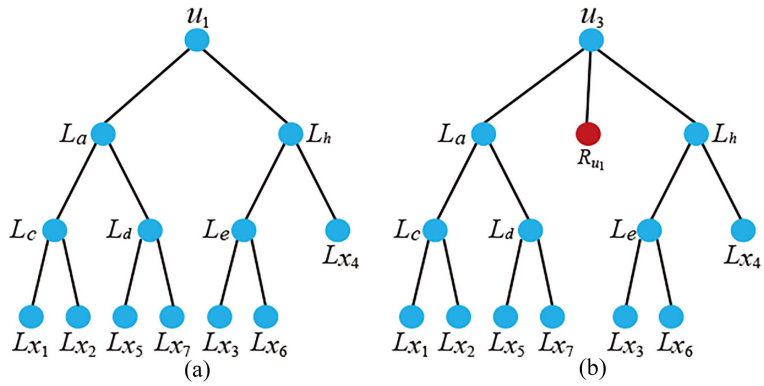

Figure 5 (Color online) Tree structures formed by different root nodes. (a) Tree with u_1 as root; (b) tree with u_3 as root.

3.2 Standard-based Tanner graph construction

After knowing the deterministic standard for SC decoding of Tanner graphs, one needs to put the standard into practical application. Specifically, search for the valid Tanner graph from the huge Tanner graph candidate space according to the standard. However, in the design of the Tanner graph, it is difficult to describe the Tanner graph directly. Instead, a Tanner graph can be indirectly characterized by check relations. For example, the Tanner graph in Figure 4 can be represented by a check relation set, which is given as

$$CheckRelation = \{26, 13, 45, 15, 67, 345, 57, 126, 367\}. \tag{26}$$

The subset $\{13\}$ represents the XOR operation of u_1 and u_3, which is described as

$$\{13\} \leftarrow u_1 \oplus u_3. \tag{27}$$

Similarly, the subset $\{13, 15, 126\}$ represents

$$\{13, 15, 126\} \leftarrow u_1 \oplus u_3 \oplus u_5 \oplus u_2 \oplus u_6, \tag{28}$$

where the arrow represents equivalent conversion. It can be found that Eq. (28) is equivalent to the operation structure on the line connecting the Tanner graph nodes u_1 and x_1 in Figure 4. Therefore, it can be safely said that the check relation set and the Tanner graph can be converted to each other.

Next, given the size and EE, the kernel and check relation set that satisfy SC decoding in (19) are constructed. Two problems need to be solved. First, given a candidate check relation set, how to quickly determine whether it can satisfy (19) based on Proposition 1. Second, continuously generate a kernel matrix and check relation set candidates. For problem 1, we first replace the check relation to the Tanner graph with the conversion from the

Algorithm 1 `BasicSearch`$(row, column, CheckRelation)$.

Require: row, $column$, $CheckRelation$;
Ensure: $WormFoot$, $newrow$, $newcolumn$;
Initial: $WormFoot = \emptyset$, $z = 0$;
$NodeMatrix =$ `TransToMatrix`$(CheckRelation)$;
$\mathcal{A} =$ `RetIns`$(row, column, NodeMatrix)$;
for $j = 1$ to $|\mathcal{A}|$ **do**
  $\mathcal{B} =$ `SearchRow`$(row, a_j, NodeMatrix)$;
  $flag = 0$;
  for $k = 1$ to $|\mathcal{B}|$ **do**
    $c_k =$ `FormCheck`(row, b_k);
    $w =$ `FindPosition`$(CheckRelation, c_k)$;
    if $c_k \in CheckRelation$ and $a_j = w$ **then**
      $WormFoot = \{WormFoot, b_k\}$; $z = z + 1$; $flag = 1$; break;
    else
      $WormFoot = \{WormFoot\}$; $z = z$; $flag = 0$;
    end if
  end for
  if $flag = 1$ **then**
    $newrow(z) = b_k$; $newcolumn(z) = a_j$;
  else
    $newrow(z) = \emptyset$; $newcolumn(z) = \emptyset$;
  end if
end for

check relation to the node matrix. In this way, the search process of the set $\mathbf{I}_{u_j}$ $(j = 1, 2, \cdots, p)$ corresponding to the root node u_j in the Tanner graph is converted to a search in the node matrix. In particular, the set $\mathbf{I}_{u_j}$ is obtained through the traces of the worm in the node matrix. In Algorithm 1, the basic search process of the worm is summarized. Given the initial position of the worm in the node matrix, which is determined by row and column, the trace at the next moment is found through the functions `TransToMatrix`, `RetIns`, `SearchRow`, `FormCheck`, and `FindPosition`.

`TransToMatrix`. Given a check relation set $CheckRelation$, convert it into a node matrix of dimension $p \times (|CheckRelation| + 1)$, where $|CheckRelation|$ is the number of elements in $CheckRelation$. Specifically, the i-th element in $CheckRelation$ represents that lines a and b in the Tanner graph form an XOR structure, corresponding to the elements with indices (a, i), $(a, i + 1)$, (b, i), and $(b, i + 1)$ in the node matrix being 1, which is expressed as

$$\begin{cases} NodeMatrix(a, i) = 1, \\ NodeMatrix(a, i + 1) = 1, \\ NodeMatrix(b, i) = 1, \\ NodeMatrix(b, i + 1) = 1. \end{cases} \tag{29}$$

Taking the $CheckRelation$ in (26) as an example, the $NodeMatrix_{7\times10}$ is initialized to a matrix of all zeros. For the first element $\{26\}$, it means that lines 2 and 6 in the Tanner graph form an XOR structure. Therefore, the elements with indexes $(2, 1)$, $(2, 2)$, $(6, 1)$, and $(6, 2)$ in $NodeMatrix_{7\times10}$ are 1. Perform similar operations on other elements in $CheckRelation$ to obtain the final $NodeMatrix_{7\times10}$:

$$NodeMatrix_{7\times10} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{30}$$

`RetIns`. Give a matrix $\boldsymbol{M}$ of size $x \times y$, row index i, $i \in \{1, 2, \cdots, x\}$, and column index j, $j \in \{1, 2, \cdots, y - 2\}$. The function `RetIns`$(i, j, \boldsymbol{M})$ outputs the column index of the first 1 in the i-th row of the matrix $\boldsymbol{M}$ that satisfies both the value greater than j and the adjacent elements are all 1.

$$\texttt{RetIns}(i, j, \boldsymbol{M}) = \{z | \boldsymbol{M}(i, z) = 1, \boldsymbol{M}(i, z + 1) = 1, z > j\}. \tag{31}$$

`SearchRow`. Give a matrix $\boldsymbol{M}$ of size $x \times y$, row index i, $i \in \{1, 2, \cdots, x\}$, and column index j, $j \in \{1, 2, \cdots, y-1\}$. The function `SearchRow`$(i, j, \boldsymbol{M})$ outputs all row indices in the matrix $\boldsymbol{M}$ where the column indexes j and $j + 1$

are both 1, except for the i-th row.

$$\texttt{SearchRow}(i, j, \boldsymbol{M}) = \{r | \boldsymbol{M}(r, j) = 1, \boldsymbol{M}(r, j+1) = 1, r \neq i\}. \tag{32}$$

`FormCheck`. Given a matrix $\boldsymbol{M}$ of size $x \times y$, i and j are the row indices of $\boldsymbol{M}$, where $i, j \in \{1, 2, \cdots, x\}$, $i \neq j$. Function $\texttt{FormCheck}(i, j)$ outputs the check relation formed by i and j:

$$\texttt{FormCheck}(i, j) = i \times 10^{size(j)} + j. \tag{33}$$

The operation $size(a)$ is the number of digits in the integer a. For example, if $a = 12$, then $size(a) = 2$.

`FindPosition`. Give a set $\boldsymbol{Q}$ with no repeated elements, and an integer b. If element b belongs to $\boldsymbol{Q}$, then the function $\texttt{FindPosition}(\boldsymbol{Q}, b)$ returns the index of the position of b in $\boldsymbol{Q}$; otherwise the output is empty.

$$\texttt{FindPosition}(\boldsymbol{Q}, b) = \begin{cases} \emptyset, & b \in \boldsymbol{Q}, \\ \{j | \boldsymbol{Q}(j) = b\}, & \text{otherwise.} \end{cases} \tag{34}$$

Taking the $NodeMatrix$ in (30) as an example, given the current position, explain Algorithm 1 to obtain the trace of the worm during the instruction set period. Assume that the row and column indexes of the worm are $i = 1$ and $j = 0$, respectively. After the function $\texttt{ReturnIns}(i, j, \boldsymbol{M})$, the instruction set $\mathcal{A} = \{2, 3, 4, 8\}$ is obtained. For the first element $a_1 = 2$ in $\mathcal{A}$, after $\texttt{SearchRow}(i, a_1, \boldsymbol{M})$, the row index 3 is returned, which means that the worm passes through the third row of the $NodeMatrix$ under the guidance of instruction a_1, and records the trace $WormFoot = \{3\}$. Next, continue to execute instruction $a_2 = 3$, and after $SearchRow(i, a_2, \boldsymbol{M})$, return the row index set $\mathcal{B} = \{4, 5\}$. Apply $\texttt{FormCheck}(i, b_k)$ to each element b_k in $\mathcal{B}$ to form the check relations set $\mathcal{C} = \{14, 15\}$. It can be found that element $c_1 = \{14\}$ does not belong to the $CheckRelation$ in (26). Although $c_2 = \{15\}$ does, the position index returned after $\texttt{FindPosition}(CheckRelation, 15)$ is $w = 4$. Since $w \neq a_2$, which implies that instruction a_2 is invalid, the trace of the worm remains unchanged at the third row. In a similar process, for instructions $a_3 = 4$ and $a_4 = 8$, the worm passes through the 5-th and 2-th rows of the $NodeMatrix$, respectively. At this point, the footprint of the worm is updated to $WormFoot = \{3, 5, 2\}$.

With row index $i \in \{1, 2, \cdots, p\}$ and column index $j = 0$ as the initial position, according to the `BasicSearch`, continuously update instructions to the worm until it traverses the entire $NodeMatrix$ and obtains the set $WormFoot$. At this point, the set $\mathbf{I}_{u_i}$ corresponding to the u_i-th root node can be obtained. If for any $\mathbf{I}_{u_i}, i = \{1, 2, \cdots, p\}$, Eq. (16) holds, then the $CheckRelation$ is legal and can be decoded according to (19). The complete search process is summarized in Algorithm 2.

For problem 2, given the kernel size and the required EE, it is necessary to continuously generate differentiated check relations for search in Algorithm 2. Function `TannerConst` in Algorithm 3 summarizes the generalized procedure of the proposed Tanner graph construction method, where function `KernelConst` can be referred to in [13]. Given kernel size p and EE, `KernelConst` continuously constructs different kernel matrices $\mathbf{F}_{p\times p}$ with EE through selection, crossover, and mutation of the generic algorithm (GA). Specifically, `KernelConst` is defined as

$$\{\mathbf{F}_{p\times p}\} = \texttt{KernelConst}(p, EE). \tag{35}$$

Each time $\mathbf{F}_{p\times p}$ with the required EE is generated. All possible check relation spaces $\mathcal{Q}$ are obtained [13]. Apply Algorithm 2 to each check relation in Q and return those satisfying (19).

3.3 Decoding algorithm

In general, the TGA-SC decoding of large kernel polar codes relies on the transfer of LLR and partialsum between adjacent stages in the Tanner graph. Consequently, the crucial step to TGA-SC decoding is to calculate the LLR and partialsum information of the kernel Tanner graph. Specifically, the kernel Tanner graph is divided into S layers from right to left; each layer contains p nodes. There are a total of $p(S-1)$ nodes in the kernel Tanner graph except for its own input and output, whose LLR and partialsum are initialized to 0. After the rightmost nodes of the kernel Tanner graph receive the channel LLR, TGA-SC loops in the order of f function, c function, g function, c function, e function, and g function until each leftmost node is estimated bit by bit. It is worth noting that the large kernel polar codes are not decoded bit by bit in ascending order, but are related to the actual designed kernel Tanner graph.

Function f calculates the LLR of the upper left node a in an XOR structure and is defined as

$$LLR_a = f(LLR_c, LLR_d) = LLR_c \boxplus LLR_d, \tag{36}$$

Algorithm 2 *Search*(*CheckRelation*).

Require: *CheckRelation*;
Ensure: *FinalCheck*;
Initial: $dyrow = \emptyset$, $dycolumn = \emptyset$; record = 0;
$NodeMatrix$ = `TransToMatrix`(*CheckRelation*);
for $i = 1$ to p **do**
  $row = i$; $column = 0$; $WormFoot = \emptyset$; $\boldsymbol{t} = \emptyset$;
  $\{nr, nc, WormFoot\} \leftarrow$ `BasicSearch`$(row, column, CheckRelation)$;
  for $z = 1$ to $|nr|$ **do**
    if `RetIns` $(nr(z), nc(z), NodeMatrix) = \emptyset$ **then**
      $\boldsymbol{t} \leftarrow \boldsymbol{t} \cup z$;
    else
      $\boldsymbol{t} = \boldsymbol{t}$;
    end if
  end for
  $nr \leftarrow nr \backslash nr(\boldsymbol{t})$; $nc \leftarrow nc \backslash nc(\boldsymbol{t})$;
  while $nr \neq \emptyset$ **do**
    for $j = 1$ to $|nr|$ **do**
      $\{nr1, nc1, WF\} \leftarrow$ `BasicSearch`$(nr(j), nc(j), CheckRelation)$; $dyrow \leftarrow \{dyrow, nr1\}$;
      $dycolumn \leftarrow \{dycolumn, nc1\}$; $WormFoot \leftarrow \{WormFoot, WF\}$;
    end for
    $nr = dyrow$; $nc = dycolumn$; $\boldsymbol{h} = \emptyset$;
    for $o = 1$ to $|nr|$ **do**
      if `RetIns` $(nr(o), nc(o), NodeMatrix) = \emptyset$ **then**
        $\boldsymbol{h} \leftarrow \boldsymbol{h} \cup o$;
      else
        $\boldsymbol{h} = \boldsymbol{h}$;
      end if
      $nr \leftarrow nr \backslash nr(\boldsymbol{h})$; $nc \leftarrow nc \backslash nc(\boldsymbol{h})$;
    end for
  end while
  if $WormFoot$ satisfies (16) **then**
    record = record + 1;
  else
    record = record;
  end if
  if record $= p$ **then**
    $FinalCheck = CheckRelation$;
  else
    $FinalCheck = \emptyset$;
  end if
end for

Algorithm 3 `TannerConst`(p, EE).

Require: p, EE;
Ensure: $\mathbf{F}_{p\times p}$, *FinalCheck*;
for $i = 1$ to N_{iter} **do**
  $\mathbf{K}_{p\times p} \leftarrow$ `KernelConst`(p, EE);
  $\mathcal{Q} \leftarrow$ obtain all check relations for $\mathbf{K}_{p\times p}$;
  for $j = 1$ to $|\mathcal{Q}|$ **do**
    $CheckRelation \leftarrow$ `Search`(Q_j);
    if $CheckRelation \neq \emptyset$ **then**
      $\mathbf{F}_{p\times p} = \mathbf{K}_{p\times p}$; $FinalCheck = CheckRelation$; break;
    else
      $\mathbf{F}_{p\times p} = \emptyset$; $FinalCheck = \emptyset$;
    end if
  end for
end for

where c and d are the upper right and lower right nodes in an XOR. The $\boxplus$ operation is calculated as

$$\sum_{i=1}^{z} \boxplus w_i = 2\tanh^{-1}\left(\prod_{i=1}^{z} \tanh\frac{w_i}{2}\right) \approx \underset{1\leqslant i\leqslant z}{\operatorname{argmin}}(|w_i|) \cdot \prod_{i=1}^{z} \operatorname{sgn}(w_i). \tag{37}$$

The c function makes a hard decision on the node's LLR and obtains the decoding result, which is defined as

$$c(LLR_i) = \begin{cases} 0, & i \notin \mathcal{I}, \\ 1, & LLR_i < 0, i \in \mathcal{I}, \\ 0, & LLR_i \geqslant 0, i \in \mathcal{I}. \end{cases} \tag{38}$$

Function g calculates the LLR of the lower left node b in an XOR structure, and is expressed as

$$LLR_b = g(LLR_c, LLR_d, c(LLR_a)) = (-1)^{c(LLR_a)} \cdot LLR_c + LLR_d. \tag{39}$$

The e function calculates the partialsum of nodes c and d, and is expressed as

$$e(c(LLR_a), c(LLR_b), i) = \begin{cases} c(LLR_a) \oplus c(LLR_b), & i = 0, \\ c(LLR_b), & i = 1. \end{cases} \tag{40}$$

Specifically, the partial sums ps_c and ps_d of nodes c and d are calculated as

$$\begin{cases} ps_c = e(c(LLR_a), c(LLR_b), 0), \\ ps_d = e(c(LLR_a), c(LLR_b), 1). \end{cases} \tag{41}$$

For further explanation, we use Figure 4 to illustrate SC decoding of the kernel Tanner graph. In particular, the kernel Tanner graph is divided into 5 layers, and the LLR of the rightmost i-th node is L_{x_i} $(i = 1, 2, \cdots, 7)$. Calculate the LLR of all nodes in the s-th layer that can perform f function from right to left. For $s = 1$, one can obtain the LLR values of nodes c and e, respectively, which are calculated as

$$LLR_c^{(1)} = f(L_{x_1}, L_{x_2}), \tag{42}$$

$$LLR_e^{(1)} = f(L_{x_3}, L_{x_6}). \tag{43}$$

For $s = 2$, $LLR_h^{(2)}$ of node h is obtained from nodes e and x_4, and $LLR_d^{(2)}$ is calculated from x_5 and x_7.

$$LLR_h^{(2)} = f(LLR_e^{(1)}, L_{x_4}) = f(f(L_{x_3}, L_{x_6}), L_{x_4}), \tag{44}$$

$$LLR_d^{(2)} = f(L_{x_5}, L_{x_7}). \tag{45}$$

For $s = 3$, only the following $LLR_a^{(3)}$ of node a can be obtained:

$$LLR_a^{(3)} = f(LLR_c^{(1)}, LLR_d^{(2)}) = f(L_{x_1} \boxplus L_{x_2}, L_{x_5} \boxplus L_{x_7}). \tag{46}$$

In a similar process, the $LLR_k^{(4)}$ of node k when $s = 4$ is computed as

$$LLR_k^{(4)} = f(LLR_a^{(3)}, LLR_h^{(2)}) = f(f(L_{x_1} \boxplus L_{x_2}, L_{x_5} \boxplus L_{x_7}), f(L_{x_3} \boxplus L_{x_6}, L_{x_4})). \tag{47}$$

Since the LLR value of u_1 is $L_{u_1} = LLR_k^{(4)}$, by applying the c function to L_{u_1}, one obtains the decoded estimate value $\hat{u}_1$ of node u_1. According to the loop constraint, the next step is to perform g operation to calculate the LLR of all nodes that can perform (39) when $s = 4$, and $LLR_m^{(4)}$ is calculated as

$$LLR_m^{(4)} = g(LLR_a^{(3)}, LLR_h^{(2)}, ps_{u_1}^{(5)}) = (-1)^{\hat{u}_1} f(L_{x_1} \boxplus L_{x_2}, L_{x_5} \boxplus L_{x_7}) + f(L_{x_3} \boxplus L_{x_6}, L_{x_4}). \tag{48}$$

Following the loop constraint, execute function $c(LLR_m^{(4)})$ to get the estimated $\hat{u}_3$ of node u_3. Then, calculate the partialsum of all nodes in each layer that can perform (41) from left to right. Specifically, according to $ps_{u_1}^{(1)}$ and $ps_{u_3}^{(1)}$, the partialsum value $ps_a^{(3)} = \hat{u}_1 \oplus \hat{u}_3$ of node a in layer $s = 3$ and $ps_h^{(2)} = \hat{u}_3$ of node h in layer $s = 2$ are obtained through e operation. Furthermore, the LLR values of nodes n and q are computed by (39) as

$$LLR_n^{(2)} = g(LLR_e^{(1)}, L_{x_4}, ps_h^{(2)}) = (-1)^{\hat{u}_3} \cdot L_{x_3} \boxplus L_{x_6} + L_{x_4}, \tag{49}$$

$$LLR_q^{(3)} = g(LLR_c^{(1)}, LLR_d^{(2)}, ps_a^{(3)}) = (-1)^{\hat{u}_1 \oplus \hat{u}_3} \cdot L_{x_1} \boxplus L_{x_2} + L_{x_5} \boxplus L_{x_7}. \tag{50}$$

Until now, one loop is completed. Continue this loop and calculate the LLR of nodes u_4, u_5, u_2, u_6, and u_7 bit by bit as

$$L_{u_4} = f((-1)^{\hat{u}_1 \oplus \hat{u}_3} \cdot L_{x_3} \boxplus L_{x_6} + L_{x_4}, (-1)^{\hat{u}_1 \oplus \hat{u}_3} \cdot L_{x_1} \boxplus L_{x_2} + L_{x_5} \boxplus L_{x_7}), \tag{51}$$

$$L_{u_5} = (-1)^{\hat{u}_4} \cdot ((-1)^{\hat{u}_1 \oplus \hat{u}_3} \cdot L_{x_3} \boxplus L_{x_6} + L_{x_4}) + (-1)^{\hat{u}_1 \oplus \hat{u}_3} \cdot L_{x_1} \boxplus L_{x_2} + L_{x_5} \boxplus L_{x_7}, \tag{52}$$

$$L_{u_2} = f((-1)^{\hat{u}_1 \oplus \hat{u}_3 \oplus \hat{u}_5} \cdot L_{x_1} + L_{x_2}, f((-1)^{\hat{u}_1 \oplus \hat{u}_3 \oplus \hat{u}_4 \oplus \hat{u}_5} \cdot L_{x_3} + L_{x_6}, (-1)^{\hat{u}_1 \oplus \hat{u}_3 \oplus \hat{u}_5} \cdot L_{x_5} + L_{x_7})), \tag{53}$$

$$L_{u_6} = (-1)^{\hat{u}_2}((-1)^{\hat{u}_1\oplus\hat{u}_3\oplus\hat{u}_5}L_{x_1} + L_{x_2}) + f((-1)^{\hat{u}_1\oplus\hat{u}_3\oplus\hat{u}_4\oplus\hat{u}_5}\cdot L_{x_3} + L_{x_6}, (-1)^{\hat{u}_1\oplus\hat{u}_3\oplus\hat{u}_5}\cdot L_{x_5} + L_{x_7}), \tag{54}$$

$$L_{u_7} = (-1)^{\hat{u}_2\oplus\hat{u}_6}((-1)^{\hat{u}_1\oplus\hat{u}_3\oplus\hat{u}_4\oplus\hat{u}_5}L_{x_3} + L_{x_6}) + (-1)^{\hat{u}_1\oplus\hat{u}_3\oplus\hat{u}_5}L_{x_5} + L_{x_7}. \tag{55}$$

Applying c function to L_{u_i} gives the decoded estimated of u_i. It can be found that the serial decoding order of the kernel Tanner graph corresponding to Figure 4 is $(u_1, u_3, u_4, u_5, u_2, u_6, u_7)$.

Given the kernel Tanner graph, the L_{u_i} of u_i to be decoded can be obtained through the functions loop process mentioned above, and the decoding order is recorded as $\mathcal{R}$. For ease of expression, the calculation of L_{u_i} $(i = \mathcal{R})$ is represented by the function $h_p^{(i)}$ $(i = \mathcal{R})$ in (19). For example, L_{u_2} in (53) can be re-expressed as $L_{u_2} = h_p^{(2)}(\hat{u}_1, \hat{u}_3, \hat{u}_4, \hat{u}_5, L_{x_1}, L_{x_2}, L_{x_3}, L_{x_5}, L_{x_6}, L_{x_7})$. Based on the decoding of the kernel Tanner graph, one completed the large kernel polar SC decoding by continuously looping the functions h_p, `KernelDecoding`, and `KernelEncoding`. In particular, the SC decoding function `KernelDecoding` of the kernel Tanner graph is defined as

$$\texttt{KernelDecoding}(h_p) = \{c(h_p^{(i)}|i = \mathcal{R})\}. \tag{56}$$

The function `KernelEncoding` calculates the partialsum of the kernel matrix $\mathbf{F}_{p\times p}$ and is defined as

$$\texttt{KernelEncoding}(u_1^p) = u_1^p\mathbf{F}_{p\times p}, \tag{57}$$

where $u_1^p \in \{0,1\}^p$. The number of initialization stages in code structure is $P = \log_p(N)$, and the decoding order $\mathcal{T}$ is defined as

$$\begin{aligned}\mathcal{T} = ((r_1-1)p + \mathcal{R}, \cdots, (r_p-1)p + \mathcal{R}, p^2 + (r_1-1)p + \mathcal{R},\\ \cdots, p^2 + (r_p-1)p + \mathcal{R}, \cdots, (N/p^2-1)p^2 + (r_p-1)p + \mathcal{R}),\end{aligned} \tag{58}$$

where r_i $(i = 1, 2, \cdots, p)$ is the i-th element in $\mathcal{R}$, and $x + \mathcal{R} = (x + r_1, x + r_2, \cdots, x + r_p)$.

TGA-SC decoding starts from the LLR at the output of the first stage of the code structure, and calculates the kernel Tanner graph that can execute the function $h_p^{(j)}$ in the i-th $(i = 1, 2, \cdots, P)$ stage to the left, where j is the updated node of the input LLR value $LLR_{input}^{(i)}$ in the i-th stage. When reaching the P-th stage, record the value of t, and then apply (56). In this way, the bits with index $T_{(t-1)p}^{tp}$ are estimated as

$$\hat{u}_{T_{(t-1)p}^{tp}} = \texttt{KernelDecoding}(LLR_{input}^{(P)}(T_{(t-1)p}^{tp})), \tag{59}$$

where $T_{(t-1)p}^{tp} = \{T_{(t-1)p}, T_{(t-1)p+1}, \cdots, T_{tp}\}$, and T_i is the i-th element in $\mathcal{T}$. The estimated decoding bits are assigned to the partialsums of the corresponding nodes in the P stage. According to the loop rule, the next step starts from the partialsum $ps_{input}^{(P)}$ at the input of the P stage of the code structure, and calculates the kernel Tanner graph that can execute (57) in the j-th $(j = P, P-1, \cdots, a)$ stage, where $1 \leqslant a \leqslant P$. The function loop process is applied continuously until all decoded bits are estimated.

The SCL decoder in classic polar holds the most likely decoding paths in a list of size L. When decoding a frozen bit, each path in the list is appended with 0 for this frozen bit. When decoding information or an unfrozen bit, each path in the list is split into two paths, and 0 and 1 are appended, respectively. Then, these $2L$ paths are compared and the L most likely paths are kept in the list. The path with the least penalty will be selected as the decoding output at the last bit. Similar ideas can also be used for TGA-SC. Therefore, TGA-SCL decoding is easily generated.

4 Enhanced TGA-SCL decoding

While employing a high-dimensional kernel with superior EE improves the polarization performance of polar codes at finite lengths, it is only when the kernel approaches infinity that full polarization is attainable [7]. Therefore, as long as the kernel dimension is not infinite, large kernel polar codes always have insufficiently polarized bit channels. PCC polar code makes full use of partially incompletely polarized bit channels to transmit parity check bits [17]. The dispersion, forward, and enhancement characteristics of the parity check bits greatly improve the error-correction capability of the classic polar codes. Drawing on the idea of PCC polar codes, this section proposes parity-check-assisted TGA-SCL (PTGA-SCL) decoding for large kernel polar codes.

In PTGA-SCL decoding, we assume that the code length is N, the number of information bits is K, and the $M+K$ bit channels with the lowest error probability are called non-frozen bit channels $\mathcal{N}$, which are used to transmit

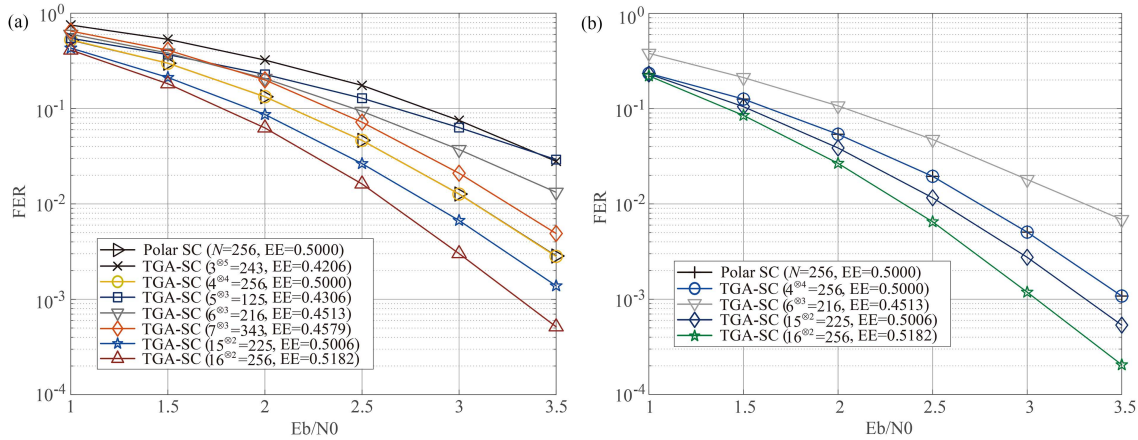

Figure 6 (Color online) FER performance of the proposed decoder under different kernel sizes. (a) Compared with classic polar SC when code rate $R = 1/2$; (b) compared with classic polar SC when code rate $R = 1/4$.

parity check and information bits. M is the number of check bits, and their positions $\mathcal{P}$ are evenly distributed in $\mathcal{N}$ and is represented as

$$\mathcal{P} = \{\mathcal{N}(i) | i = \lfloor j \cdot (K + M)/M \rfloor, j = 1, 2, \cdots, M\}, \tag{60}$$

where $\lfloor\rfloor$ is a round-down operation. The M check bits are randomly connected to the previous information bits to form a check relationship, which is defined as

$$x_{\mathcal{P}_m} = \left(\sum_{j \in \mathcal{E}_m} x_j \right) \bmod 2, \tag{61}$$

where $\mathcal{P}_m$ and $x_{\mathcal{P}_m}$ $(m = 1, 2, \cdots, M)$ are the position and value of the m-th parity check bit, respectively, and $\mathcal{E}_m$ is the information bits index set that forms a check relationship with the m-th parity check bit.

The difference from TGA-SCL decoding is that PTGA-SCL supplements the decoding of parity check bits. In particular, parity check bits rely on (61) for bit estimation instead of LLR. When the parity check bit is estimated by (61) and LLRs are inconsistent, a penalty will be imposed on the current decoding path, thereby reducing the survival probability of its subpath.

5 Simulation results

In this section, numerical results are implemented to evaluate the performance of the devised TGA-SC decoder. We assume BPSK transmission with $\{+1, -1\}$ over AWGN channel. All code rates $R = N/K$ are 1/2 and the frame error rate (FER) is used as a criterion to measure decoding performance.

5.1 Error-correction performance

Figure 6 portrays the FER performance of TGA-SC decoding for large kernel polar codes under different kernel sizes, with classic polar SC decoding serving as a benchmark. Specifically, Figure 6(a) shows the performance when the code rate $R = 1/2$. As can be seen, the error-correction performance of TGA-SC is almost determined by the error exponent of the kernel matrix. Specifically, for kernel $\mathbf{F}_{15}$, although the EE is slightly greater than $\mathbf{F}_4$, it achieves a significant improvement in FER performance. Simulations show that the error-correction capability of the TGA-SC exceeds classic polar SC decoding when EE is greater than 0.5. More specifically, when the kernel size $l = 16$, code length $N = 256$, and $FER = 10^{-2}$, the devised TGA-SC decoding obtains a gain of about 0.4 dB compared with the polar SC method. To present further insights, Figure 6(b) compares the FER performance of TGA-SC when the code rate $R = 1/4$. The phenomenon mentioned above still exists. In other words, when the EE is higher than that of the classic polar code, the error-correction capability of TGA-SC decoding is significantly improved.

Figure 7 compares the FER performance of TGA-SCL with existing decoding schemes for large kernel polar codes. In particular, in Figure 7(a), the SC-based recursive-trellis [11] and W-expression [12] method is employed to serve as a benchmark. It can be seen that the FER performance of TGA-SCL is similar to that of W-expression under

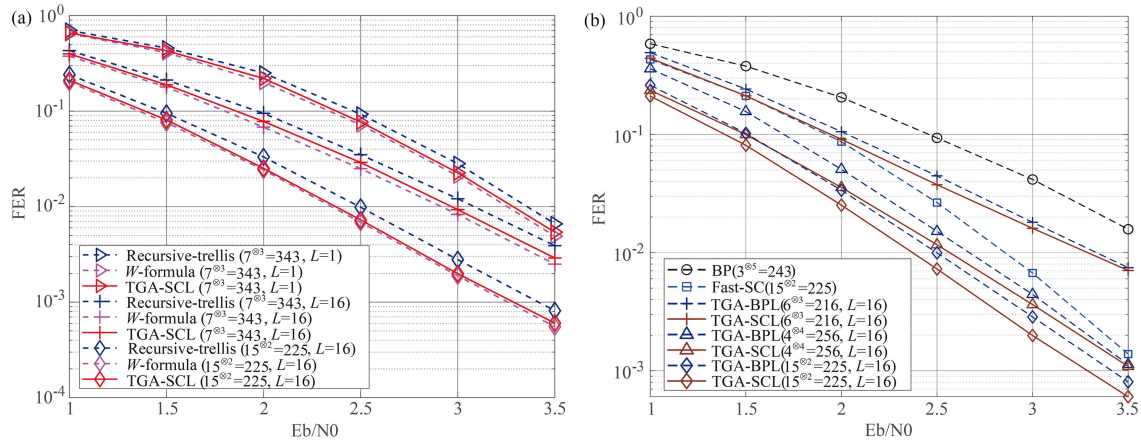

Figure 7 (Color online) Proposed TGA-SCL decoding performance for large kernels. (a) Compared with SC-based W-expression and recursive-trellis decoding; (b) compared with parallel BP and fast decoding.

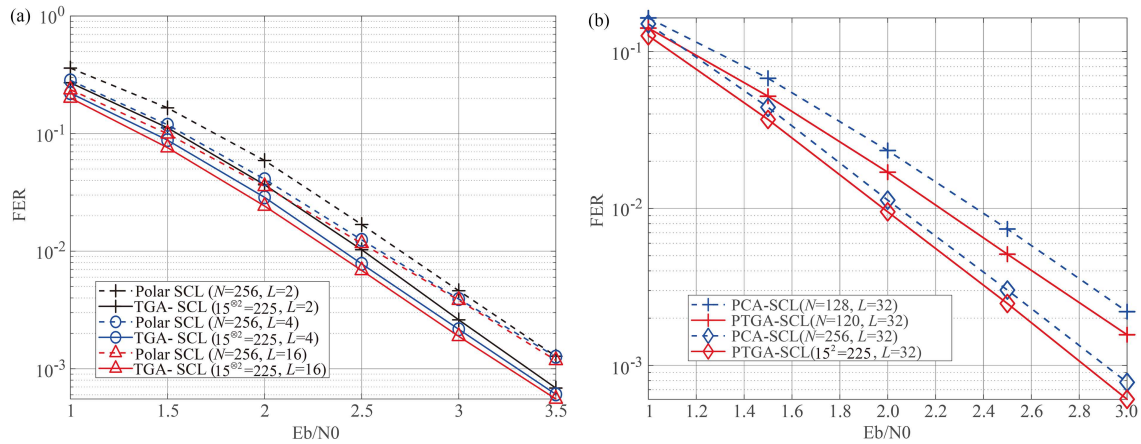

Figure 8 (Color online) FER performance of TGA-SCL decoding. (a) Compared with polar SCL decoding; (b) compared with PCC polar code.

different kernel sizes, and both outperform recursive-trellis. In addition, the complexity of TGA-SCL is much lower than that of the W-expression, which is explained in Subsection 5.2. Figure 7(b) demonstrates the performance of the TGA-SCL with recently proposed parallel TGA-BPL decoding [13]. Note that the TGA-BPL decoder employs an early termination criterion and has a complexity of $O(LT \cdot \frac{2QN}{p} \cdot \log_p N)$, where Q is the number of XORs in the kernel Tanner graph. We observe that the invented TGA-SCL decoding has similar complexity to TGA-BPL while achieving significant improvement in error correction capability. Specifically, when kernel size $p = 15$, code length $N = 225$, and FER $= 10^{-3}$, the TGA-SCL decoding obtains a gain of about 0.2 dB. In addition, Figure 7(b) is also compared with the parallel BP in [18] and the fast decoding in [6], and significant improvements in error correction capability are achieved.

Figure 8(a) compares the FER performance of TGA-SCL decoding with classic polar SCL decoding. The simulations confirm that the proposed decoding methods achieve more significant error-correction performance gains than polar SCL. For example, when the code length $N = 225$, $L = 16$, and FER$= 10^{-2}$, the devised TGA-SCL decoding obtains a gain of about 0.3 dB compared with polar SCL. Figure 8 shows a comparison between PTGA-SCL and parity-check-aided SCL (PCA-SCL) decoding of the 5G standard PCC polar [1]. We show simulation results for $N = 225$, $N = 120$, and all code rates are 1/2. In particular, for code length $N = 120$, PTGA-SCL adopts the form of a hybrid kernel, and the generator $\mathbf{G}_N = \mathbf{F}_{15} \otimes \mathbf{F}_2^{\otimes 3}$. The number of check bits is set to $M = 8$ for both decoders. The results show that PTGA-SCL outperforms PCA-SCL under different code lengths.

5.2 Complexity analysis

For TGA-SC with code length N, its decoding is based on the offline optimized kernel Tanner graph. Therefore, the complexity of TGA-SC decoding mainly depends on the calculation of node LLR in the Tanner graph. Specifically,

Table 1 Complexity comparison of different decoding schemes.

Polar (SC)	Polar (SCL)	Large kernel (straightforward SC)	W-expression	TGA-BPL	TGA-SCL
$O(N\log_2 N)$	$O(LN\log_2 N)$	$O(2^p LN\log_p N)$	$O(p^2 N\log_p N)$	$O(LT\frac{2QN}{p}\log_p N)$	$O(L\frac{2EN}{p}\log_p N)$

the Tanner graph has $\log_p N$ stages from right to left, and each stage has N/p consistent kernel Tanner graphs. Assuming that there are E XOR structures in a single kernel Tanner graph, then the entire Tanner graph has $(NE/P)\log_p N$ XORs. Note that when all elements of the kernel matrix are 1, E reaches its maximum value p^2. Because in an XOR structure, the LLRs of the upper left and lower left nodes need to be calculated according to (36) and (39), respectively. Therefore, when the standard Landau notation $O(\cdot)$ is employed for denoting the complexity, a TGA-SC decoder for a large kernel polar code can be implemented with complexity $O(\frac{2EN}{p}\log_p N)$. When it comes to TGA-SCL decoding, since there are L paths in the decoding process, its complexity is approximately L times that of TGA-SC and can be expressed as $O(L\frac{2EN}{p}\log_p N)$.

Table 1 shows the complexity comparison of different decoding strategies for classical polar codes and large kernel polar codes. It can be found that the proposed TGA-SCL can achieve a complexity similar to that of classical polar SCL decoding, both of which are lower than the W-expression. Specifically, the proposed TGA-SCL decoding reduces the computational complexity of direct SC decoding of large kernel polar codes from exponential to linear level, which makes the practical application of large kernel polar codes possible.

6 Conclusion

In this paper, we proposed a TGA-SC decoding for large kernel polar to meet the stringent reliability and latency constraints in 6G URLLC. We establish theoretical criteria for the Tanner graph to satisfy serial decoding constraints, accompanied by rigorous proofs. Then, a gamified worm search scheme is designed, which makes the proposed criteria practical for screening Tanner graphs by accurately tracking the worm's running trajectory in the node matrix. Finally, the Tanner graph-based SC decoding for large kernel polar codes is proposed. Benefiting from the strategies mentioned above, our scheme achieves significant complexity reduction and error-correction enhancement compared to existing decoding methods.

References

1 Jiang T, Liu Y, Xiao L, et al. PCC polar codes for future wireless communications: potential applications and design guidelines. IEEE Wireless Commun, 2024, 31: 414–420
2 Wu Y Z, Li L, Fan P Z. A fast parallel SC-Fano decoding algorithm for PAC codes. Sci China Inf Sci, 2023, 66: 152301
3 Xu W, Huang Y M, Wang W, et al. Toward ubiquitous and intelligent 6G networks: from architecture to technology. Sci China Inf Sci, 2023, 66: 130300
4 Xu W, Yang Z, Ng D W K, et al. Edge learning for B5G networks with distributed signal processing: semantic communication, edge computing, and wireless sensing. IEEE J Sel Top Signal Process, 2023, 17: 9–39
5 Wang C X, You X, Gao X, et al. On the road to 6G: visions, requirements, key technologies, and testbeds. IEEE Commun Surv Tutorials, 2023, 25: 905–974
6 Ashikhmin A, Trifonov P. Fast successive cancellation decoding of polar codes with large kernels. IEEE Trans Commun, 2025, 73: 3–11
7 Korada S B, Sasoglu E, Urbanke R. Polar codes: characterization of exponent, bounds, and constructions. IEEE Trans Inform Theor, 2010, 56: 6253–6264
8 Lin H P, Lin S, Abdel-Ghaffar K A S. Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents. IEEE Trans Inform Theor, 2015, 61: 5253–5270
9 Wang X, Zhang Z, Zhang L. On the SC decoder for any polar code of length $N = l^n$. In: Proceedings of 2014 IEEE Wireless Communications and Networking Conference (WCNC), 2014. 485–489
10 Trofimiuk G, Trifonov P. Window processing of binary polarization kernels. IEEE Trans Commun, 2021, 69: 4294–4305
11 Trifonov P, Karakchieva L. Recursive processing algorithm for low complexity decoding of polar codes with large kernels. IEEE Trans Commun, 2023, 71: 5039–5050
12 Huang Z, Jiang Z, Zhou S, et al. On the non-approximate successive cancellation decoding of binary polar codes with medium kernels. IEEE Access, 2023, 11: 87505–87519
13 Liu Y Y, Zhang Y, Liu G H, et al. Tanner-graph-assisted belief propagation decoding for large kernel polar codes: low-complexity design and enhancement method. Sci China Inf Sci, 2025, 68: 212301
14 Pang Q K, Ma Z, Tang X H. Unreliability normalization weighted bit-flipping algorithms of LDPC decoding for ReRAM systems. Sci China Inf Sci, 2024, 67: 229304
15 Hu Y Y, Pan Z W, Guan Y L. Asynchronous multilevel bit-interleaved polar-coded modulation. Sci China Inf Sci, 2023, 66: 132304
16 Yu Y R, Pan Z W, Tan X S, et al. A latency-reduced successive cancellation list decoder for polar codes. Sci China Inf Sci, 2019, 62: 029302
17 Wang T, Qu D, Jiang T. Parity-check-concatenated polar codes. IEEE Commun Lett, 2016, 20: 2342–2345
18 Sha J, Liu J, Wang Z. Improved BP decoder for polar codes based on a modified kernel matrix. Electron Lett, 2016, 52: 1982–1984