Fast Successive-Cancellation-Based Decoders of Polar Codes

Maryam Haghighi Ardakani, Muhammad Hanif, Masoud Ardakani, *Senior Member, IEEE*, and Chintha Tellambura, *Fellow, IEEE*

***Abstract*— The successive-cancellation list (SCL) and successive-cancellation flip (SCF) decoding can be used to improve the performance of polar codes, especially for short to moderate length codes. However, their serial decoding nature results in significant decoding latencies. Implementing some operations in parallel can reduce their decoding latencies. This paper presents fast implementations of the SCL and SCF decoders. In particular, we propose fast parallel list decoders for five newly identified types of nodes in the decoding tree of a polar code, which significantly improves the decoding latency. We also present novel fast SCF decoders that decode some special nodes in the decoding tree of a polar code without serially computing bit log-likelihood ratios. Using our proposed fast parallel SCF decoders, we observed an improvement up to 81% with respect to the original SCF decoder. This significant reduction in the decoding latency is observed without sacrificing the bit-error-rate performance of the code.**

***Index Terms*— Polar codes, successive-cancellation decoder, list decoder, flip decoder, fast decoding, decoding latency.**

I. Introduction

POLAR codes, discovered by Arıkan, will be used in the next generation mobile communication standards [1]. This new class of error correcting codes has an explicit construction and can achieve the symmetric capacity of memoryless channels under successive-cancellation (SC) decoding as the code length goes to infinity [2]. However, the error correction performance of polar codes for short to moderate block lengths with the SC decoder is inferior to that of the other state-of-the-art error correcting codes, such as low-density parity-check (LDPC) and turbo codes. This is due to the significant performance gap between the SC and the maximum-likelihood (ML) decoders [3]. Unfortunately, ML decoding is beset with high computational complexity, even for moderate-length polar codes. Thus, the successive-cancellation list (SCL) decoder and successive-cancellation flip (SCF) decoder aim to achieve low complexity.

Among the existing low-complexity polar decoders, the SCL decoder achieves near ML performance even with moderate list sizes [3]. Aiding it with a cyclic-redundancy check (CRC) further improves the bit-error-rate (BER) performance beyond that of LDPC and turbo codes [3].

The SCL decoder, similar to the original SC decoder, operates serially as it estimates one bit at a time. However, unlike the latter which keeps only a single decoding path, the SCL decoder maintains a list of L most probable paths while decoding each bit [3].

The serial decoding nature of the SC and SCL decoders produce high decoding latencies. To shorten them, multiple schemes have been developed [4]–[13]. The main underlying idea behind these schemes is to parallelize some operations to increase the decoder throughput. In particular, these schemes recognize certain patterns in the codeword and implement their fast parallel decoders. For example, [4] identifies and proposes low-complexity parallel decoders for rate-0 and rate-1 nodes for the SC decoder. Similarly, [5] proposes fast decoders for single-parity-check (SPC) and repetition (REP) nodes in the decoding tree of a polar code. Recently, [6] presented fast low-complexity decoders of five nodes (type-I, type-II, type-III, type-IV, and type-V nodes) to further increase the decoding speed of the SC decoder.

Fast SCL decoders follow a similar strategy to reduce decoding latency. However, unlike fast SC decoders that output only the most-probable output sequence, fast SCL decoders maintain and output a list of the most probable paths. In particular, to improve the decoding speed of the SCL decoder, [10]–[12] propose list decoders for rate-0, rate-1, REP and SPC nodes.

Although the SCL decoder achieves near-ML performance, compared to the SC decoder, it has higher computational and memory complexities that grow linearly with the list size. More specifically, for a code of size N the memory and computational complexities of the SC decoder are $\mathcal{O}(N)$ and $\mathcal{O}(N \log N)$, respectively, whereas, for a list of size L, they increase to $\mathcal{O}(LN)$ and $\mathcal{O}(LN \log N)$, respectively, in the SCL decoder. Further, fast implementations based on hardware unrolling result in a quadratic increase in the hardware area [14]. These shortcomings make the fast implementation of SCL decoder challenging. As such, a different approach was introduced in [15] to improve the performance of SC decoder: SC flip (SCF) decoding.

Manuscript received July 13, 2018; revised November 8, 2018 and January 19, 2019; accepted February 25, 2019. Date of publication March 19, 2019; date of current version July 13, 2019. This work was supported by the TELUS Corporation and Natural Sciences and Engineering Research Council of Canada. This paper was presented in part at the IEEE Wireless Communications and Networking Conference (WCNC) Workshop, Barcelona, Spain, April 2018. The associate editor coordinating the review of this paper and approving it for publication was P. Trifonov. *(Corresponding author: Maryam Haghighi Ardakani.)*

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2R3, Canada (e-mail: maryam3@ualberta.ca; mhanif@ualberta.ca; ardakani@ualberta.ca; ct4@ualberta.ca).

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

Digital Object Identifier 10.1109/TCOMM.2019.2906232

The SCF decoder, unlike the SCL decoder that parallelizes multiple SC decoders, relies on multiple sequential applications of the SC decoder. In particular, the SCF decoder allows up to a given number of decoding trials. The first trial is essentially the same as SC decoding, but it also forms a list of $T_{\max}$ bit-flip positions. The bit-flip positions correspond to the information bits with the smallest absolute values of the decision log-likelihood ratios (LLRs). If the CRC is not satisfied in the initial SC decoding, then one bit is flipped in the next trial, and the standard SC decoder is used to decode the subsequent positions. A new decoding trial is launched unless the CRC is satisfied or the maximum number of trials is reached. The aforementioned procedure of the SCF decoding has $\mathcal{O}(N)$ memory complexity, and its average computational complexity is $\mathcal{O}(N \log N)$ at high signal-to-noise ratios (SNR) [15].

Similar to the SC and SCL decoders, the decoding latency of the SCF decoder can be improved by implementing fast decoders of different nodes. In particular, a fast parallel decoder of a node, in addition to outputting the codeword, should also compute and update the list of most probable bit-flip positions. Recently, [16] introduced fast SC flip decoders for the REP, rate-1, SPC, and type-I nodes. However, some decision metrics used in these decoders to compute the most-probable bit-flip positions, result in a BER performance loss compared to the original SCF decoder [15].

In this paper, we first present fast list decoders for five recently-identified nodes [6] in the decoder tree of a polar code. Implementing them along with the existing list decoders of special nodes can increase the decoder throughput significantly without sacrificing the BER performance. Furthermore, we present a fast SCF decoder. While [16] uses some decision metrics without providing the justification, we follow the approach used in [12] and [13] to generate the most likely codewords and propose new procedure to compute and update the list of most-probable bit-flip positions. We investigate rate-1, SPC and type-I nodes with our proposed procedure. Moreover, in order to further reduce the decoding latency, we adapt the existing fast SC decoders of type-II, type-III, type-IV and type-V nodes [6]. Our proposed decoder has a smaller latency and better BER performance compared to the existing fast SCF decoder.

The remainder of this paper is organized as follows: Section II provides background information on polar code construction and SC-based decoding algorithms. The fast SCL decoder of five recently-identified nodes is proposed in Section III. We next propose fast SCF decoders in Section IV. In Section V the decoding latency of proposed decoders are compared with those of the existing ones. Simulation results are presented in Section VI. Finally, we conclude the paper in Section VII.

II. Background

A binary polar code, $P(N,k)$, of length $N = 2^n$, where n is a positive integer, and rate k/N maps the input vector $\mathbf{u} = \{u_0, u_1, \cdots, u_{N-1}\}$ to the output vector

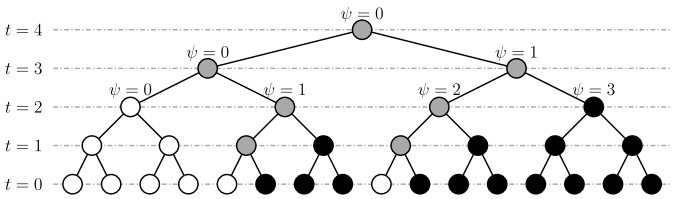

Fig. 1. Binary-tree representation of a polar code $P(16, 10)$.

$\mathbf{x} = \{x_0, x_1, \cdots, x_{N-1}\}$ such that[1]

$$\mathbf{x} = \mathbf{u}\mathbf{F}^{\otimes n}, \tag{1}$$

where $\mathbf{F}^{\otimes n}$ is the nth tensor power of $\mathbf{F}$ defined as

$$\mathbf{F}^{\otimes n} = \begin{bmatrix} \mathbf{F}^{\otimes(n-1)} & \mathbf{O} \\ \mathbf{F}^{\otimes(n-1)} & \mathbf{F}^{\otimes(n-1)} \end{bmatrix}, \tag{2}$$

with $\mathbf{F}^{\otimes 0} = 1$.

The matrix $\mathbf{F}^{\otimes n}$ synthesizes N polarized channels from N independent copies of a given channel. Amongst the N synthesized channels, the k most reliable ones are used to carry the information. We denote the index set of k information bits by $\mathcal{I}$. Furthermore, $\mathcal{F}$ is used to denote the index set of $N-k$ least reliable synthesized channels. Then, for every $i \in \mathcal{F}$ we set u_is to 0 and refer them as frozen bits.

After receiving the encoded vector $\mathbf{x}$ from the channel, the receiver provides the LLR vector of the received bits, denoted by $\mathbf{y}_{ch} = \{y_0, y_1, \cdots, y_{N-1}\}$, to the polar-code decoder. After receiving the LLRs, a polar-code decoder estimates the input and output vectors. We use $\hat{\mathbf{u}}$ and $\hat{\mathbf{x}}$ to denote the estimated input and output vectors, respectively.

A. SC, SCL and SCF Decoding

The SC, SCL and SCF decoding can be understood using a binary-tree representation of the polar code. Fig. 1 depicts a $P(16, 10)$ code in its binary-tree representation, where black and white leaf nodes are the information and frozen bits, respectively.

In Fig. 1, t denotes the level in the decoding tree and $0 \leq \psi < 2^{n-t}$ is the count from left to right at level t. $R = 2^t$ is used to denote the length of a node rooted at level t. Observe that each node in the code tree of a polar code has a bijective relationship with (ψ, t) pair. For example, the root node corresponds to $(0, n)$, whereas the ith leaf node can be represented as $(i, 0)$. Further, with the exception of leaf nodes, each node (ϕ, t) has two children: the left child $(2\phi, t-1)$ and the right child $(2\phi+1, t-1)$.

1) SC Decoding: The SC decoding can be viewed as information exchange between nodes in the decoding tree. In particular, each node (ϕ, t) receives a soft information vector, $\mathbf{y}^{\phi,t}$. The root node receives channel LLRs; i.e., $\mathbf{y}^{0,n} = \mathbf{y}_{ch}$, whereas other nodes receive $\mathbf{y}^{\phi,t}$ from their parents. The node (ϕ, t) then computes hard bit estimates $\mathbf{x}^{\phi,t}$ from $\mathbf{y}^{\phi,t}$,

[1]For the sake of exposition, we consider only non-permuted polar codes as similar conclusions can be drawn for permuted-polar codes.

and send these estimates to its parent. In particular, with the exception of the leaf nodes, each node upon receiving $\mathbf{y}^{\phi,t}$ generates LLR vectors for its children as

$$\mathbf{y}_k^{2\phi,t-1} = 2\,\text{arctanh}\left(\tanh(\frac{y_k^{\phi,t}}{2})\tanh(\frac{y_{k+2^{t-1}}^{\phi,t}}{2})\right) \quad (3)$$

$$\mathbf{y}_k^{2\phi+1,t-1} = y_{k+2^{t-1}}^{\phi,t} + (1-2\mathbf{x}_k^{2\phi,t-1})y_k^{\phi,t}, \quad (4)$$

where $0 \leq k < 2^{t-1}$ is used to denote the kth entry of a vector. On the other hand, $\mathbf{x}^{\phi,t}$ is computed as

$$\mathbf{x}_i^{\phi,t} = \begin{cases} \mathbf{x}_i^{2\phi,t-1} \oplus \mathbf{x}_i^{2\phi+1,t-1}, & \text{if } i < 2^{t-1}; \\ \mathbf{x}_{i-2^{t-1}}^{2\phi+1,t-1}, & \text{otherwise,} \end{cases} \quad (5)$$

where $\oplus$ denotes binary XOR operation, and $0 \leq i < 2^t$.

The decoding process starts from the root node, i.e., by setting $\mathbf{y}^{0,n} = \mathbf{y}_{ch}$. Each node, upon receiving its LLR vector, computes and passes LLR vectors to its children until the leaf nodes receive their LLR values. At the ith leaf node, i.e. node $(i,0)$, the ith input bit $\mathbf{u}_i$ is estimated as

$$\hat{\mathbf{u}}_i = \mathbf{x}^{i,0} = \begin{cases} 0, & \text{if } i \in \mathcal{F}; \\ H(\mathbf{y}^{i,0}), & \text{otherwise,} \end{cases} \quad (6)$$

where $H(\mathbf{y})$ makes a hard decision on each element of $\mathbf{y}$ as

$$H(y_i) = \begin{cases} 0, & \text{when } y_i \geq 0; \\ 1, & \text{otherwise.} \end{cases} \quad (7)$$

The leaf nodes then sends $\mathbf{x}^{i,0}$ to their parents. Each node (ϕ,t) upon receiving $\mathbf{x}^{2\phi,t-1}$ and $\mathbf{x}^{2\phi+1,t-1}$ from its children, computes $\mathbf{x}^{\phi,t}$ using (5). Finally, the hard decision estimate of $\mathbf{x}$ is computed as $\hat{\mathbf{x}} = \mathbf{x}^{0,t}$. The decoding process is completed when $\hat{\mathbf{x}} = \mathbf{x}^{0,n}$ is computed at the root node.

2) SCL Decoding: The SCL decoder [3] operates in a similar fashion to the SC decoder. Instead of outputting only one codeword estimate, the SCL decoder maintains a list of L candidate codewords with their corresponding path metrics (PM) while decoding each bit. In particular, the PM associated with the lth candidate codeword, PM_i^l, after estimating the ith bit is computed as [17]

$$\text{PM}_i^l = \sum_{k=0}^{i-1} \ln\left(1 + e^{-(1-2\hat{\mathbf{u}}_{k_l})\mathbf{y}^{k_l,0}}\right), \quad (8)$$

where $\hat{\mathbf{u}}_{k_l}$ is the estimate of the $(k+1)$-th bit, and $\mathbf{y}^{k_l,0}$ is the LLR received by the $(k+1)$-th leaf node in the path l.

If the $(i+1)$-th bit is a frozen bit then $\hat{\mathbf{u}}_{i_l} = 0$ for each list, and the PMs are updated accordingly. But if the $(i+1)$-th bit is an information bit, then each path generates two paths corresponding to $\hat{\mathbf{u}}_{i_l} = 0$ and $\hat{\mathbf{u}}_{i_l} = 1$. The PMs are updated accordingly, and a total of $2L$ paths are generated. Amongst them, only those L paths are maintained that have the lowest PMs.

3) SCF Decoding: The SCF decoding can be viewed as multiple SC decoding operating in a sequential manner. In the initial phase, the codeword is decoded through the standard SC decoding. In addition to the decoding, the SC decoder identifies $T_{\max}$ bit-flip positions. The bit-level LLRs, $|y^{i,0}|$, corresponding to the bit-flip positions have the smallest absolute values.

Let $\mathbf{t} = \{t_0, t_1, \ldots, t_{T_{\max}-1}\}$ denote the set of bit-flip positions, where $\mathbf{t} \subset \mathcal{I}$ and $|y^{t_i,0}| \leq |y^{t_j,0}|$ for $0 \leq i \leq j < T_{\max}$. After the initial decoding, if the estimated codeword satisfies the CRC, the decoding process stops. Otherwise, another SC decoding trial is carried out. However, in this trial $\hat{u}_{t_0}$ is flipped and the subsequent bits are decoded with the standard SC decoder. The decoder outputs the estimated codeword if it satisfies the CRC. Otherwise, a new SC decoding trial is launched. But this time $\hat{u}_{t_1}$ is flipped instead. Again the CRC is checked, and new trial is carried out if it is not satisfied. This process continues until the CRC of the estimated codeword is satisfied or the maximum number of trials $T_{\max}+1$ is reached.

B. Fast Decoding

The sequential nature of the decoding and the tree traversal from top to bottom and left to right results in a high decoding latency. The decoding speed can be improved by implementing decoders that do not traverse the decoder tree and output multiple bits in parallel. Based on this idea, researchers have proposed fast parallel decoders for different nodes that improve the decoding speed significantly [4]–[6]. For example, a rate-0 node (shown by unfilled circles in Fig. 1) can be decoded without computing the LLR values of its children by noting that all the leaf nodes of a rate-0 node correspond to frozen bits. Therefore, $\mathbf{x}_k^{\phi,t} = \mathbf{0}$. Likewise, fast decoders for rate-1, REP, SPC, type-I, type-II, type-III, type-IV, and type-V nodes improve the SC decoder throughput [4]–[6].

The decoders proposed in [4]–[6] only output a single codeword. The SCL decoder, on the other hand, maintains a list of L codewords for each bit. As such, the SC decoders of the above mentioned nodes require modifications to be used for the SCL decoder. Among the aforementioned nodes, the SCL decoders for only rate-0, rate-1, REP and SPC nodes have been proposed in [10]–[12]. To the best of authors' knowledge, the SCL decoders for the remaining five nodes are not available in the literature. Hence, we present list decoders for the remaining nodes to improve the decoding speed of the existing fast SCL decoder.

Since the SCF decoder uses the SC decoding in each trial, implementing fast SC decoders will improve the decoding latency of the SCF decoder. However, the SCF decoder must maintain a list of the most probable bit-flip positions also and should be able to flip the bits at these positions. Therefore, the existing fast SC decoders cannot be used directly to increase the decoding speed.

Observe that the existing decision metric used to determine the list of the most probable bit-flip positions is based on the bit-level LLR. But computing the bit-level LLRs through the tree traversal would result in a very high decoding latency. Therefore, we propose a decision metric that reflects the effect of the bit flip on the log likelihood of the estimated codeword. The proposed decision metric can be computed without traversing the code tree and hence, facilitates fast SCF decoding.

Fast SCF decoders for REP, rate-1, SPC, and type-I special nodes have been proposed in [16], but some of them incur BER performance loss.

In this paper, we first propose modifications of these decoders and present the fast SCF decoders for the remaining special nodes. Our proposed fast SCF decoders not only result in reduced decoding latency but also improve BER performance of the existing fast SCF decoder [16].

To better understand the proposed fast SC decoders, we find the following notations quite useful: For a node (ψ, t), we define $\tilde{A}_{\psi,t} = \{i : i \in \{\!\{R\}\!\}, \text{ and } 2^t\psi + i \in \mathcal{I}\}$, and $\tilde{A}^c_{\psi,t} = \{i : i \in \{\!\{R\}\!\}, \text{ and } 2^t\psi + i \in \mathcal{F}\}$. Here $R = 2^t$ and $\{\!\{R\}\!\}$ denotes the set $\{0, 1, \cdots, R-1\}$. With this notation, rate-0 and rate-1 nodes correspond to $\tilde{A}_{\psi,t} = \{\}$ and $\tilde{A}_{\psi,t} = \{\!\{R\}\!\}$, respectively. REP and SPC nodes are recognized when $\tilde{A}_{\psi,t} = \{R-1\}$ and $\tilde{A}^c_{\psi,t} = \{0\}$, respectively. Furthermore, type-I node corresponds, to $\tilde{A}_{\psi,t} = \{R-2, R-1\}$; type-II node, to $\tilde{A}_{\psi,t} = \{R-3, R-2, R-1\}$; type-III node, to $\tilde{A}^c_{\psi,t} = \{0, 1\}$; type-IV node, to $\tilde{A}^c_{\psi,t} = \{0, 1, 2\}$; and type-V node, to $\tilde{A}_{\psi,t} = \{R-5, R-3, R-2, R-1\}$.

III. Proposed Fast List Decoders

Before presenting the list decoders for the five nodes, we first examine the contribution of a node (ϕ, t) to the PM of a codeword. Observe that the leaf nodes corresponding to the node (ϕ, t) are the nodes $(2^t\phi + k, 0)$, where $0 \le k < 2^t$. Consequently, using (8), the contribution of the node (ϕ, t) to the PM, denoted by $\mathrm{PM}^{\phi,t}$, is given by

$$\mathrm{PM}^{\phi,t} = \sum_{k=2^t\phi}^{2^t(\phi+1)-1} \ln\left(1 + e^{-(1-2\hat{\mathbf{u}}_k)\mathbf{y}^{k,0}}\right). \tag{9}$$

Our proposed decoders, similar to the list decoders of the rate-0, rate-1, REP and SPC nodes [11], compute $\mathrm{PM}^{\phi,t}$ without traversing the whole subtree corresponding to a node. To this end, for the PM computation we use an important result introduced in [11, Th. 1], which specifically proves that[2]

$$\mathrm{PM}^{\phi,t} = \sum_{k=0}^{2^t-1} \ln\left(1 + e^{-(1-2\mathbf{x}^{\phi,t}_k)\mathbf{y}^{\phi,t}_k}\right). \tag{10}$$

Another result, that will be used extensively in our proposed decoders to simplify the calculations is the following identity.

$$\ln\left(1 + e^{a}\right) - \ln\left(1 + e^{-a}\right) = a. \tag{11}$$

In the following, we present fast decoders for the type-I, type-II, type-III, type-IV and type-V nodes. For better readability, we use $R = 2^t$ and drop the indexes ϕ and t from the notations. Further, we add l, where $0 \le l < L$, in the notations to indicate that the calculations pertain to the path l.

A. Type-I Node

For a type-I node, $\mathbf{x}^l = \{\mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}}, \cdots, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}}\}$ [6]. As such, $\mathbf{x}^l$ equals only one of the four codewords: $C_{00} = \{0, 0, \cdots, 0, 0\}$, $C_{01} = \{0, 1, \cdots, 0, 1\}$, $C_{10} = \{1, 0, \cdots, 1, 0\}$, and $C_{11} = \{1, 1, \cdots, 1, 1\}$.

In the proposed decoder, we compute the PM contribution of the type-I node corresponding to each codeword for each list. In particular, the PMs corresponding to the four codewords for the list l, $0 \le l < L$, are

$$\mathrm{PM}^l_{00} = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{-\mathbf{y}^l_{2k}}\right) + \ln\left(1+e^{-\mathbf{y}^l_{2k+1}}\right), \tag{12}$$

$$\mathrm{PM}^l_{01} = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{-\mathbf{y}^l_{2k}}\right) + \ln\left(1+e^{+\mathbf{y}^l_{2k+1}}\right), \tag{13}$$

$$\mathrm{PM}^l_{10} = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{+\mathbf{y}^l_{2k}}\right) + \ln\left(1+e^{-\mathbf{y}^l_{2k+1}}\right), \tag{14}$$

$$\mathrm{PM}^l_{11} = \mathrm{PM}^l + \sum_{k=0}^{R/2-1} \ln\left(1+e^{+\mathbf{y}^l_{2k}}\right) + \ln\left(1+e^{+\mathbf{y}^l_{2k+1}}\right). \tag{15}$$

Using (11), the calculations can be simplified as

$$\mathrm{PM}^l_{00} = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-\mathbf{y}^l_k}\right), \tag{16}$$

$$\mathrm{PM}^l_{01} = \mathrm{PM}^l_{00} + \varsigma_1, \tag{17}$$

$$\mathrm{PM}^l_{10} = \mathrm{PM}^l_{00} + \varsigma_0, \tag{18}$$

$$\mathrm{PM}^l_{11} = \mathrm{PM}^l_{00} + \varsigma_0 + \varsigma_1, \tag{19}$$

where $\varsigma_i = \sum_{k=0}^{R/2-1} \mathbf{y}^l_{2k+i}$ for $i = 0, 1$.

As a result of these calculations, a total of $4L$ path metrics are computed. Amongst them, only the smallest L ones are retained, and the corresponding codewords are assigned to $\mathbf{x}^l$.

B. Type-II Node

For a type-II node, $\mathbf{x}^l = \{\mathbf{x}_{\text{R-4}}, \mathbf{x}_{\text{R-3}}, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}}, \cdots, \mathbf{x}_{\text{R-4}}, \mathbf{x}_{\text{R-3}}, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}}\}$, where $\mathbf{x}_{\text{R-4}} = \mathbf{x}_{\text{R-3}} \oplus \mathbf{x}_{\text{R-2}} \oplus \mathbf{x}_{\text{R-1}}$ [6]. Hence, $\mathbf{x}^l$ can only be one of the eight codewords: $C_{000} = \{0, 0, 0, 0 \cdots, 0, 0, 0, 0\}$, $C_{001} = \{1, 0, 0, 1, \cdots, 1, 0, 0, 1\}$, $C_{010} = \{1, 0, 1, 0, \cdots, 1, 0, 1, 0\}$, $C_{011} = \{0, 0, 1, 1, \cdots, 0, 0, 1, 1\}$, $C_{100} = \{1, 1, 0, 0, \cdots, 1, 1, 0, 0\}$, $C_{101} = \{0, 1, 0, 1, \cdots, 0, 1, 0, 1\}$, $C_{110} = \{0, 1, 1, 0 \cdots, 0, 1, 1, 0\}$, and $C_{111} = \{1, 1, 1, 1 \cdots, 1, 1, 1, 1\}$.

Following the approach used in the type-I list decoder, we compute

$$\mathrm{PM}^l_{000} = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-\mathbf{y}^l_k}\right), \tag{20}$$

$$\mathrm{PM}^l_{001} = \mathrm{PM}^l_{000} + \eta_0 + \eta_3, \tag{21}$$

$$\mathrm{PM}^l_{010} = \mathrm{PM}^l_{000} + \eta_0 + \eta_2, \tag{22}$$

$$\mathrm{PM}^l_{011} = \mathrm{PM}^l_{000} + \eta_2 + \eta_3, \tag{23}$$

$$\mathrm{PM}^l_{100} = \mathrm{PM}^l_{000} + \eta_0 + \eta_1, \tag{24}$$

$$\mathrm{PM}^l_{101} = \mathrm{PM}^l_{000} + \eta_1 + \eta_3, \tag{25}$$

$$\mathrm{PM}^l_{110} = \mathrm{PM}^l_{000} + \eta_1 + \eta_2, \tag{26}$$

$$\mathrm{PM}^l_{111} = \mathrm{PM}^l_{000} + \eta_0 + \eta_1 + \eta_2 + \eta_3. \tag{27}$$

where $\eta_i = \sum_{k=0}^{R/4-1} \mathbf{y}^l_{4k+i}$ for $0 \le i < 4$.

Afterwards, L PMs are retained amongst the $8L$ computed PMs, and the corresponding codewords are assigned to $\mathbf{x}^l$.

[2] Although [11] presented the equivalence of PMs for a rate-1 node, the same result is valid for any node and can be proved using a similar approach.

C. Type-III Node

A type-III node corresponds to $\hat{A}^c = \{0, 1\}$; i.e., there are only two frozen bits in the node [6]. Therefore, $\mathbf{x}^l$ can be one of the $2^{R-2} = 2^{2^t-2}$ valid type-III node codewords. Since the size of a type-III node can be large, it is impractical to compute a total of $2^{R-2}L$ path metrics and choose the best L ones amongst them. Hence, we follow the approach used in [10] and [12] of generating candidate codewords.

For each path l, we first find the ML codeword and compute its corresponding PM. Since the even-indexed and odd-indexed bits of a type-III codeword constitute two separate SPC codes [6], we compute the ML codeword by using Wagner decoding [18]. In particular, we set $\mathbf{x}^l_k = H(\mathbf{y}^l_k)$, where $H(y)$ is defined in (7), and $0 \leq k < R$. We then find the location of the least-reliable even-indexed and odd-indexed LLR values as

$$(e_l, o_l) = (\underset{0\leq k<R/2}{\operatorname{argmin}} |\mathbf{y}^l_{2k}|, \underset{0\leq k<R/2}{\operatorname{argmin}} |\mathbf{y}^l_{2k+1}|) \tag{28}$$

and compute the parities of both SPC codes as

$$(\gamma^l_e, \gamma^l_o) = \left(\bigoplus_{k=0}^{R/2-1} \mathbf{x}^l_{2k}, \bigoplus_{k=0}^{R/2-1} \mathbf{x}^l_{2k+1} \right). \tag{29}$$

Further, we set $\mathbf{x}^l_{2e_l} = \mathbf{x}^l_{2e_l} \oplus \gamma^l_e$ and $\mathbf{x}^l_{2o_l+1} = \mathbf{x}^l_{2o_l+1} \oplus \gamma^l_o$ to satisfy the even-parity constraint. Lastly, we compute the PM corresponding to the ML codeword as

$$\mathrm{PM}^l_{\mathrm{ML}} = \Delta^l + \gamma^l_e |\mathbf{y}^l_{2e_l}| + \gamma^l_o |\mathbf{y}^l_{2o_l+1}|, \tag{30}$$

where $\Delta^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-|\mathbf{y}^l_k|}\right)$.

We then define modified LLRs, α^l_i's, as $\alpha^l_{2k} = |\mathbf{y}^l_{2k}| + (1 - 2\gamma^l_e)|\mathbf{y}^l_{2e_l}|$, and $\alpha^l_{2k+1} = |\mathbf{y}^l_{2k+1}| + (1 - 2\gamma^l_o)|\mathbf{y}^l_{2o_l+1}|$ for $0 \leq k < R/2$. Next, we sort the modified LLRs (excluding $\alpha^l_{2e_l}$ and $\alpha^l_{2o_l+1}$) in an ascending order and denote the sorted indexes by $(i)_l$, where $0 \leq i < R-2$. Mathematically, $\alpha^l_{(k)_l} \leq \alpha^l_{(k+1)_l}$ for $0 \leq k < R-3$.

After computing the sorted indexes for each path, we start generating candidate codewords by flipping bits of the ML codeword. In particular, starting from $k = 0$, we generate two codewords corresponding to the bit $(k)_l$ being 0 or 1. In order to satisfy the even parity condition, the bits with indexes $2e^l$ and $2o^l + 1$ are modified accordingly. The PMs corresponding to the generated codewords are calculated as

$$\mathrm{PM}^l_{(k)_l} = \begin{cases} \mathrm{PM}^l_{(k-1)_l}, & \text{if } \mathbf{x}^l_{(k)_l} = H(\mathbf{y}^l_{(k)_l}); \\ \mathrm{PM}^l_{(k-1)_l} + |\mathbf{y}^l_{(k)_l}| + \beta_{(k)_l}, & \text{otherwise,} \end{cases} \tag{31}$$

where $\mathrm{PM}^l_{(-1)_l} = \mathrm{PM}^l_{\mathrm{ML}}$, $\beta_{(k)_l} = (1 - 2\gamma^l_{e,k-1})|\mathbf{y}^l_{2e_l}|$ if $(k)_l$ is even, and $\beta_{(k)_l} = (1 - 2\gamma^l_{o,k-1})|\mathbf{y}^l_{2o_l+1}|$ otherwise. The bit $\gamma^l_{e,k}$ equals $\gamma^l_{e,k-1}$ when $(k)_l$ is odd. When $(k)_l$ is even then

$$\gamma^l_{e,k} = \begin{cases} \gamma^l_{e,k-1}, & \text{if } \mathbf{x}^l_{(k)_l} = H(\mathbf{y}^l_{(k)_l}); \\ 1 - \gamma^l_{e,k-1}, & \text{otherwise.} \end{cases} \tag{32}$$

Likewise, $\gamma^l_{o,k} = \gamma^l_{o,k-1}$ when $(k)_l$ is even. When $(k)_l$ is odd, $\gamma^l_{o,k}$ is updated as

$$\gamma^l_{o,k} = \begin{cases} \gamma^l_{o,k-1}, & \text{if } \mathbf{x}^l_{(k)_l} = H(\mathbf{y}^l_{(k)_l}); \\ 1 - \gamma^l_{o,k-1}, & \text{otherwise.} \end{cases} \tag{33}$$

Here, $\gamma^l_{e,-1} = \gamma^l_e$, and $\gamma^l_{o,-1} = \gamma^l_o$.

As a result of the aforementioned operations, a total of $2L$ PMs are computed for each k. Amongst them, only the lowest L ones are retained. We continue this process of creating $2L$ codewords from the existing L ones and retaining only L best ones in a successive manner as k varies from 0 to $\min\{L-2, R-3\}$. After that, we return the L surviving paths and their PMs. We do not consider all the codewords; i.e., we do not vary k from 0 to $R-3$ when $L < R-1$, because Theorem 1 asserts that the extra codewords created after $k = L-2$ are not included in the surviving paths.

Theorem 1: In the proposed list decoder of the type-III node, the codewords that have $\mathbf{x}^l_{(k)_l} \neq H(\mathbf{y}^l_{(k)_l})$ for $k \geq L-1$ are not amongst the surviving codewords.

Proof: We prove this theorem by contradiction. Suppose the codeword with $\mathbf{x}^l_{(L-1)_l} \neq H(\mathbf{y}^l_{(L-1)_l})$ is among the surviving codewords. Consequently, the PM corresponding to such a codeword is one of the smallest L ones. Note that the minimum possible value of the corresponding PM is $\mathrm{PM}^l_{\mathrm{ML}} + \alpha^l_{(L-1)_l}$, which represents the case that all the bits of the generated codeword and the ML codewords are the same except the bit $(L-1)_l$ and the corresponding parity bit.

Now consider the codewords corresponding to $\mathbf{x}^l_{(k)_l} = H(\mathbf{y}^l_{(k)_l})$ and $\mathbf{x}^l_{(k)_l} = 1 - H(\mathbf{y}^l_{(k)_l})$ for $0 \leq k < L-1$. Amongst them, we consider the following L codewords. One of them is the ML codeword with the path metric of $\mathrm{PM}^l_{\mathrm{ML}}$. The other $L-1$ codewords are the ones that differ from the ML codeword only in two locations: the $(k)_l$ bit and the corresponding (even/odd) parity bit. The path metric of such codewords are $\mathrm{PM}^l_{\mathrm{ML}} + \alpha^l_{(k)_l}$ for $0 \leq k < L-1$. Using the fact that $\alpha^l_i \geq 0$ and $\alpha^l_{(k)_l} \leq \alpha^l_{(L-1)_l}$, the PMs of the considered L codewords is less than or equal to $\mathrm{PM}^l_{\mathrm{ML}} + \alpha^L_{(L-1)_l}$. Consequently, the PM of the codeword corresponding to $\mathbf{x}^l_{(L-1)_l} \neq H(\mathbf{y}^l_{(L-1)_l})$ is not amongst the best L ones. Therefore, we do not need to consider multiple codewords corresponding to $\mathbf{x}^l_{(L-1)_l}$ being equal to 0 or 1. Rather, we just set $\mathbf{x}^l_{(L-1)_l} = H(\mathbf{y}^l_{(L-1)_l})$.

Similar assertions can be made for $\mathbf{x}^l_{(k)_l}$ for $k \geq L$. ■

D. Type-IV Node

The codeword in a type-IV node satisfies the following relationship [6].

$$\sum_{i=0}^{R/4-1} x_{4i} = \sum_{i=0}^{R/4-1} x_{4i+1} = \sum_{i=0}^{R/4-1} x_{4i+2} = \sum_{i=0}^{R/4-1} x_{4i+3} = z, \tag{34}$$

where z can be 0 or 1.

In the proposed list decoder of a type-IV node, we first compute the ML codewords for each list corresponding to $z = 0$ and $z = 1$ using Wagner decoder. In particular, we compute the i^*_0, i^*_1, i^*_2, and i^*_3 as

$$i^*_j = \underset{0\leq k<R/4}{\operatorname{argmin}} |\mathbf{y}^l_{4k+j}|, \tag{35}$$

where $j = 0, 1, 2, 3$.

Then we set $\mathbf{x}_k^l = H(\mathbf{y}_k^l)$ and compute γ_j^l as

$$\gamma_j^l = \bigoplus_{k=0}^{R/4-1} \mathbf{x}_{4k+j}^l, \tag{36}$$

where $j = 0, 1, 2, 3$. Afterwards, we generate two ML codewords corresponding to $z = 0$ and $z = 1$ by setting $\mathbf{x}_{4i_j^*+j}^l = \mathbf{x}_{4i_j^*+j}^l \oplus \gamma_j^l \oplus z$. Their corresponding PMs are

$$\mathrm{PM}_{\mathrm{ML},z}^l = \Delta^l + \sum_{j=0}^{3} (\gamma_j^l \oplus z)|\mathbf{y}_{4i_j^*+j}^l|, \tag{37}$$

where $\Delta^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-|\mathbf{y}_k^l|}\right)$.

As a result, we compute $2L$ codewords and their corresponding PMs. Amongst them, we only retain those L codewords that have the smallest PM values. Afterwards, using z of the retained codewords, we update γ_j^l as $\gamma_j^l = \gamma_j^l \oplus z^l$.

Similar to the type-III node decoder, we then define α_i's as $\alpha_{4k+j}^l = |\mathbf{y}_{4k+j}^l| + (1 - 2\gamma_j^l)|\mathbf{y}_{4i_j^*+j}^l|$ for $0 \le k < R/4$, and $0 \le j \le 3$. Next, excluding $\alpha_{4i_j^*+j}^l$, we sort α_i^l's in an ascending order and denote the sorted indexes by $(i)_l$, where $0 \le i < R-4$. Thus, $\alpha_{(k)_l}^l \le \alpha_{(k+1)_l}^l$ for $0 \le k < R-5$.

Starting from $k = 0$, we then compute different codewords and their corresponding PMs by considering $\mathbf{x}_{(k)_l} = 0$ and $\mathbf{x}_{(k)_l} = 1$ and setting appropriate values to the $\mathbf{x}_{4i_j^*+j}^l$, where $j = [\![(k)_l]\!]_2$ is the remainder after division of $(k)_l$ by $2^2 = 4$. The PMs are computed as

$$\mathrm{PM}_{(k)_l}^l = \begin{cases} \mathrm{PM}_{(k-1)_l}^l, & \text{if } \mathbf{x}_{(k)_l}^l = H(\mathbf{y}_{(k)_l}^l); \\ \mathrm{PM}_{(k-1)_l}^l + |\mathbf{y}_{(k)_l}^l| + \beta_{(k)_l}, & \text{otherwise,} \end{cases} \tag{38}$$

where $\mathrm{PM}_{(-1)_l}^l = \mathrm{PM}_{\mathrm{ML},z}^l$ and $\beta_{(k)_l} = (1 - 2\gamma_{j,k-1}^l)|\mathbf{y}_{4i_j^*+j}^l|$. Here, $\gamma_{j,k}^l$ is computed recursively as

$$\gamma_{j,k}^l = \begin{cases} \gamma_{j,k-1}^l, & \text{if } \mathbf{x}_{(k)_l} = H(\mathbf{y}_{(k)_l}); \\ 1 - \gamma_{j,k-1}^l, & \text{otherwise,} \end{cases} \tag{39}$$

with $\gamma_{j,-1}^l = \gamma_j^l$. Also, $\gamma_{i,k}^l = \gamma_{i,k-1}^l$ for $0 \le i < 4$ and $i \ne j$, and $\gamma_{i,-1}^l = \gamma_i^l$.

The aforementioned procedure results in a total of $2L$ codewords for each k. We then save only those codewords and their corresponding PMs that have the lowest PM values. We continue this process till k reaches $\min\{L-2, R-5\}$. If $L < R-3$, we simply set $\mathbf{x}_{(k)_l}^l = H(\mathbf{y}_{(k)_l}^l)$ for $L-2 < k \le R-5$ as the codeword with $\mathbf{x}_{(k)_l}^l \ne H(\mathbf{y}_{(k)_l}^l)$ has a larger PM and is not amongst the surviving codewords.[3]

E. Type-V Node

The list decoder for a type-V node is quite similar to that of the type-I and type-II nodes. In particular,

$$\mathbf{x}^l = \mathbf{m}\begin{bmatrix}\mathbf{G}_0 & \cdots & \mathbf{G}_0\end{bmatrix}, \tag{40}$$

where $\mathbf{m}$ is a binary row vector of length 4, and

$$\mathbf{G}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{41}$$

Similar to the proposed decoders for type-I and type-II nodes, we compute 16 PMs corresponding to different values of $\mathbf{m}$ as

$$\mathrm{PM}_{0000}^l = \mathrm{PM}^l + \sum_{k=0}^{R-1} \ln\left(1 + e^{-\mathbf{y}_k^l}\right), \tag{42}$$

$$\mathrm{PM}_{0001}^l = \mathrm{PM}_{0000}^l + \sigma_1 + \sigma_2 + \sigma_4 + \sigma_7, \tag{43}$$

$$\mathrm{PM}_{0010}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_2 + \sigma_4 + \sigma_6, \tag{44}$$

$$\mathrm{PM}_{0011}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_1 + \sigma_6 + \sigma_7, \tag{45}$$

$$\mathrm{PM}_{0100}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_1 + \sigma_4 + \sigma_5, \tag{46}$$

$$\mathrm{PM}_{0101}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_2 + \sigma_5 + \sigma_7, \tag{47}$$

$$\mathrm{PM}_{0110}^l = \mathrm{PM}_{0000}^l + \sigma_1 + \sigma_2 + \sigma_5 + \sigma_6, \tag{48}$$

$$\mathrm{PM}_{0111}^l = \mathrm{PM}_{0000}^l + \sigma_4 + \sigma_5 + \sigma_6 + \sigma_7, \tag{49}$$

$$\mathrm{PM}_{1000}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_1 + \sigma_2 + \sigma_3, \tag{50}$$

$$\mathrm{PM}_{1001}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_3 + \sigma_4 + \sigma_7, \tag{51}$$

$$\mathrm{PM}_{1010}^l = \mathrm{PM}_{0000}^l + \sigma_1 + \sigma_3 + \sigma_4 + \sigma_6, \tag{52}$$

$$\mathrm{PM}_{1011}^l = \mathrm{PM}_{0000}^l + \sigma_2 + \sigma_3 + \sigma_6 + \sigma_7, \tag{53}$$

$$\mathrm{PM}_{1100}^l = \mathrm{PM}_{0000}^l + \sigma_2 + \sigma_3 + \sigma_4 + \sigma_5, \tag{54}$$

$$\mathrm{PM}_{1101}^l = \mathrm{PM}_{0000}^l + \sigma_1 + \sigma_3 + \sigma_5 + \sigma_7, \tag{55}$$

$$\mathrm{PM}_{1110}^l = \mathrm{PM}_{0000}^l + \sigma_0 + \sigma_3 + \sigma_5 + \sigma_6, \tag{56}$$

$$\mathrm{PM}_{1111}^l = \mathrm{PM}_{0000}^l + \sum_{i=1}^{7} \sigma_i, \tag{57}$$

where $\sigma_i = \sum_{k=0}^{R/8-1} \mathbf{y}_{8k+i}^l$ for $0 \le i < 8$. Amongst the $16L$ computed path metrics, only the smallest L ones are kept, and their corresponding codewords are assigned to $\mathbf{x}^l$.

Remark 1: In the aforementioned list decoders, we did not provide calculations for $\hat{\mathbf{u}}_k^l$, where $2^t\phi \le k < 2^t(\phi+1)$ for a node (ϕ, t). For systematic polar codes, we do not compute $\hat{\mathbf{u}}^l$ as the information bits appear transparently in $\mathbf{x}$ [19]. For non-systematic polar codes, we can use $\hat{\mathbf{u}}^{\phi,t} = \mathbf{x}^{\phi,t}\mathbf{F}^{\otimes t}$ to find the information bits corresponding to the codeword $\mathbf{x}^{\phi,t}$. Note that the above operation, in contrast to the conventional list decoding, involves only bit operations, which can be carried out expeditiously with very few hardware resources.

For the type-I, type-II and type-V nodes, we can further reduce the hardware complexity and decoding latency by using the special structure of $\mathbf{x}^{\phi,t}$ and the frozen-bit pattern. For example, for a type-I node, $\hat{\mathbf{u}}$ can be calculated as follows: $\hat{\mathbf{u}}_k = 0$ for $2^t\phi \le k < 2^t(\phi+1) - 2$, $\hat{\mathbf{u}}_{2^t(\phi+1)-2} = \mathbf{x}_{2^t-2}^{\phi,t} \oplus \mathbf{x}_{2^t-1}^{\phi,t}$, and $\hat{\mathbf{u}}_{2^t(\phi+1)-1} = \mathbf{x}_{2^t-1}^{\phi,t}$.

Remark 2: The computational and hardware complexity of PM calculations can be reduced using [17]

$$\ln(1 + e^a) \approx \begin{cases} a, & \text{if } a > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{58}$$

[3]The proof of this assertion is quite similar to that of Theorem 1. We skip the proof due to the repetition of ideas.

For example, using (58)*, we can compute* PM_{00}^{l} *for a type-I node as*

$$\mathrm{PM}_{00}^{l} \approx \mathrm{PM}^{l} + \sum_{k:\mathbf{y}_k^l<0} |\mathbf{y}_k^l|. \tag{59}$$

IV. Proposed Fast Flip Decoders

SCF decoding is another polar code decoding scheme that outperforms SC decoding. Similar to the SCL decoding, the SCF decoder selects the most-probable codeword amongst multiple codewords. However, unlike the SCL decoder, the memory requirements of the SCF decoder do not scale linearly with the number of codewords considered. Furthermore, the computational complexity of the SCF decoder is similar to that of the SC decoder when the channel has high reliability. As such, the SCF decoding has gained interest in the research community recently [16], [20], [21].

The SCF decoder is basically a modified SC decoder that runs up to a maximum of $T_{\max}+1$ times one after the other. In the first trial, in addition to performing the SC decoding, the SCF decoder maintains a list of $T_{\max}$ bit-flip positions. If the initial SC decoding fails, then one bit (whose location is in the maintained list) is flipped in the next trial, and the standard SC decoder is used to decode the subsequent positions. The decoding process terminates if the CRC is satisfied, otherwise a new decoding trial is initiated until the maximum number of trials is reached.

The sequential decoding nature of the SC decoder results in high decoding latency of the SCF decoder. Furthermore, fast SC decoders of [4]–[9] cannot be used directly to increase the speed because the SCF decoder needs to maintain the list of the least-reliable bit locations. Specifically, the conventional SCF decoder [15] uses the bit-level LLR, $|y^{i,0}|$ to determine the bit-flip locations. Whereas, the existing fast SC decoder of a node (ϕ,t) does not compute the bit-level LLRs to improve the throughput. Since computing the bit-level LLRs will incur a high decoding latency, we propose a decision metric to determine bit-flip locations that can be computed without traversing the decoding tree. In the following, we first introduce the decision metric, and then we describe the fast SCF decoders for the special nodes based on the proposed metric.

A. Proposed Decision Metric

Before we present our proposed decision metric, we first analyze the decision metric used in the conventional SCF decoder, i.e., $|y^{i,0}|$ [15]. To this end, we consider the path metric of a codeword decoded by the conventional list decoder for $L=1$.

The conventional SCL decoder considers two possibilities for the bit u_i, i.e., 0 or 1, and selects the path with the least PM. Using (8), it can be verified that the difference between the PMs of the considered codewords is exactly $|y^{i,0}|$. That is, the decision metric used in the conventional SCF decoder equals the difference between the PMs of the codewords that share the same code bits except the code bits corresponding to the node $(i,0)$.

We use the aforementioned observation to propose a decision metric for our fast SCF decoders. Since our decoders decode the codewords corresponding to a node without traversing it, we devise a decision metric that can be computed without the traversal. In particular, we propose to use the following decision metric:

$$\lambda = \mathrm{PM}_{\mathbf{x}} - \mathrm{PM}_{\mathrm{ML}}. \tag{60}$$

Here, $\mathrm{PM}_{\mathbf{x}}$ and $\mathrm{PM}_{\mathrm{ML}}$ are the contributions of a codeword $\mathbf{x}$ and the ML codeword to the PM, respectively. The ML codeword is the codeword that is selected by the SCL decoders for $L=1$, i.e., it has the least associated PM. Observe that our proposed metric λ coincides with $|y^{i,0}|$ for the nodes $(i,0)$, which is used in the conventional SCF decoding [15].

It should be noted that [16] also proposed decision metrics for some special nodes. Some of these metrics, however, result in a performance degradation. Our proposed decoders, on the other hand, are based on the contribution of the selected codeword to the PM and, hence, show better BER performance than the decoders of [16].

B. Proposed Fast Decoders

In the following, we first proposed SCF decoders of REP, rate-1, SPC and type-I nodes. We also compare them with the proposed SCF decoders in [16] highlighting the key differences, which eventually result in the improvement of the BER performance. Afterwards, we present fast SCF decoders for the remaining nodes (type-II, type-III, type-IV, and type-V nodes).

C. REP Node

A REP node contains only one information bit. Therefore, there are two valid REP node codewords: an all-zeros and an all-ones codewords. The PM contributions for these codewords can be computed as

$$\mathrm{PM}_0 = \sum_{k=0}^{R-1} \ln\left(1+e^{-\mathbf{y}_k}\right), \tag{61}$$

and

$$\mathrm{PM}_1 = \sum_{k=0}^{R-1} \ln\left(1+e^{\mathbf{y}_k}\right). \tag{62}$$

When the ML codeword is an all-zeros codeword, $\mathrm{PM}_{\mathrm{ML}} = \mathrm{PM}_0$, and $\lambda = \mathrm{PM}_1 - \mathrm{PM}_0$; otherwise $\lambda = \mathrm{PM}_0 - \mathrm{PM}_1$. Equivalently, $\lambda = |\mathrm{PM}_1 - \mathrm{PM}_0|$, which can be simplified using (11) as

$$\lambda = \left|\sum_{k=0}^{R-1} y_k\right|. \tag{63}$$

Note that [16] also used the same decision metric for a REP node.

The decision metric in (63) is equivalent to that of the conventional SCF decoder [15] computed for the only non frozen bit in $\mathbf{u}$ supported by a REP node.

After the first trial, in case the bit-flip position belongs to a REP node, all bits of the estimated ML codeword, $\mathbf{x}^{ML}$, of the

REP node are flipped. This process is equivalent to flipping process in [15], i.e., flipping the only non frozen bit of the REP node at $\mathbf{u}$.

D. Rate-1 Node

The ML codeword of a rate-1 node is obtained as $\mathbf{x}_k^{\mathrm{ML}} = H(\mathbf{y}_k)$ for $0 \leq k < R$, and its corresponding PM is

$$\mathrm{PM}_{\mathrm{ML}} = \sum_{k=0}^{R-1} \ln\left(1 + e^{-|\mathbf{y}_k|}\right). \tag{64}$$

The proposed SCF decoder for a rate-1 node follows the approach used in its fast SCL decoder [12] to compute the contributions of other codewords to the PM. In particular, codewords are generated by flipping bits, individually and simultaneously, relative to the ML codeword as is done in the decoding of a type-III node. Specifically, the absolute values of LLRs are sorted. We denote the sorted indexes by (i), where $0 \leq i < R$. Next, PMs are computed in a successive manner by path splitting for each index (i). Since, we are generating $T_{\max}$ numbers of the most likely codewords, as we do in the SCL decoder with list size L [12, Th. 1], we only consider path splitting for $0 \leq i < \min\{T_{\max}, R-1\}$. This limits the number of search which is required to find bit-flip positions. The contributions of the rate-1 node codewords can be obtained as

$$\mathrm{PM}_{\mathbf{x}} = \mathrm{PM}_{\mathrm{ML}} + \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|. \tag{65}$$

Here, $\mathbf{f}_{\mathbf{x}}$ is the set of those indexes of codeword $\mathbf{x}$ at which the code bits differ that of the ML codeword. Consequently, the decision metric can be computed as

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|. \tag{66}$$

Observe that [16] introduced $\lambda_{\mathbf{x}} = |y_{(i)}|$ for $0 \leq i \leq R-1$ as the decision metric for a rate-1 node. This metric computes the contribution of those codewords that differ from the ML codeword only in one bit location. For example, the decision metric used in [16] would compute $\lambda_2 = |y_{(2)}|$. On the other hand, we compute $\lambda_2 = |y_{(2)}|$ when $|y_{(2)}| < |y_{(0)}| + |y_{(1)}|$, and $\lambda_2 = |y_{(0)}| + |y_{(1)}|$ otherwise.

In the decoding trials following the initial one, in case the bit-flip positions belong to a rate-1 node, we flip the bits in $\mathbf{f}_{\mathbf{x}}$ of the estimated ML codeword.

E. SPC Node

The ML codeword of an SPC node can be estimated using Wagner decoding [18]. In particular, we set $\mathbf{x}_k = H(\mathbf{y}_k)$ for $0 \leq k < R$. We then find the location of the least-reliable LLR value and compute the parity as

$$p = \underset{0 \leq k < R}{\operatorname{argmin}} |\mathbf{y}_k|, \tag{67}$$

and

$$\gamma_{\mathrm{ML}} = \bigoplus_{k=0}^{R-1} \mathbf{x}_k, \tag{68}$$

respectively. Further, we set $\mathbf{x}_p = \mathbf{x}_p \oplus \gamma_{\mathrm{ML}}$ to satisfy the even-parity constraint. The PM corresponding to the ML codeword is given by

$$\mathrm{PM}_{\mathrm{ML}} = \Delta + \gamma_{\mathrm{ML}} |\mathbf{y}_p|, \tag{69}$$

where $\Delta = \sum_{k=0}^{R-1} \ln\left(1 + e^{-|\mathbf{y}_k|}\right)$.

In our proposed decoder, the absolute values of LLRs (excluding y_p) are sorted in an ascending order. We use (i), where $0 \leq i < R-1$ to denote the sorted indexes. Next, we generate codewords by flipping bits relative to the ML codeword such that the parity constraint is satisfied. This can be done by 'path splitting' as described in [12] and in the proposed SCL decoders for the Type-III and Type-IV nodes.

The contribution of the generated codewords to the PM can be shown to be

$$\mathrm{PM}_{\mathbf{x}} = \Delta + \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|. \tag{70}$$

Here, $\mathbf{f}_{\mathbf{x}}$ is the set of indexes of the codeword $\mathbf{x}$ which differ from $H(\mathbf{y})$, where $H(\mathbf{y})$ is defined in (7). Consequently, the decision metrics for the node can be computed as

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j| - \gamma_{\mathrm{ML}} |y_p|. \tag{71}$$

Note that [16] introduced the following metric

$$\lambda_{\mathbf{x}} = |y_{(i)}| + (-1)^{\gamma_{\mathrm{ML}}} |y_p|, \tag{72}$$

which corresponds to only double bit flips relative to the ML codeword. Since (72) does not consider some valid codewords that have more occurrence likelihood than the considered codewords, it results in a BER performance loss compared to the conventional SCF decoder. Our proposed decoders, on the other hand, do not incur any BER performance loss compared to the conventional SCF decoder, as also evident from the simulation results presented in Section VI.

If a trial of the fast SCF decoder needs to flip bits in an SPC node, the bits stored in $\mathbf{f}_{\mathbf{x}}$ are flipped relative to the hard-decision bit estimates.

F. Type-I Node

A Type-I node contains two information bits, and its codeword is $\mathbf{x} = \{x_{R-2}, x_{R-1}, \ldots, x_{R-2}, x_{R-1}\}$. Thus, the even and odd indexed code bits constitute two repetition codes.

The proposed SCF decoder computes the PMs corresponding to $(x_{R-2}, x_{R-1}) = (0,0), (0,1), (1,0), (1,1)$ as mentioned in Section III-A and selects the one which has the least PM as the ML codeword. Next, it computes the decision metrics corresponding to other codewords using (60). It can be verified that the decision metric corresponding to the codeword that has all of its even-indexed bits flipped with respect to the ML codeword is

$$\lambda_0 = |\varsigma_0|. \tag{73}$$

Likewise, the decision metric corresponding to the codeword that differs the ML codeword at odd-indexed bits is

$$\lambda_1 = |\varsigma_1|, \tag{74}$$

and the codeword that differs from the ML codeword at all locations has the following decision metric:

$$\lambda_2 = |\varsigma_0| + |\varsigma_1|. \tag{75}$$

The conventional SCF decoder considers only two additional codewords: the codeword with the second last and last bits in the $\mathbf{u}$ vector flipped relative to the decoded SC codeword. These codewords correspond to the codewords with decision metrics (74) and (75), respectively. The type-I node decoder presented in [16] only computes the codewords with decision metrics λ_0 and λ_1; i.e., it ignores a valid conventional SCF codeword. Our proposed decoder, on the other hand, computes the decision metric for both codewords. In addition, it considers a valid type-I codeword that has both of its bits in $\mathbf{u}$ flipped relative to that of the ML codeword (codeword with decision metric (73)). This codeword has more likelihood (and hence has less contribution to the PM) than the codeword that has its last bit in $\mathbf{u}$ flipped.

G. Type-II Node

A type-II node contains three information bits and its codeword is $\mathbf{x} = \{\mathbf{x}_{\text{R-4}}, \mathbf{x}_{\text{R-3}}, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}}, \ldots, \mathbf{x}_{\text{R-4}}, \mathbf{x}_{\text{R-3}}, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}}\}$, where $\mathbf{x}_{\text{R-4}} = \mathbf{x}_{\text{R-3}} \oplus \mathbf{x}_{\text{R-2}} \oplus \mathbf{x}_{\text{R-1}}$ [6]. Hence, $(\mathbf{x}_{\text{R-3}}, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}})$ can only be one of the eight combinations: (000), (001), (010), (011), (100), (101), (110), and (111).

The proposed decoders for the type-II node works similary to that of a type-I node. In particular, we first compute 8 PM values corresponding to the valid codewords of a type-II node. Next, the ML codeword is found by selecting the code with the least PM. Then, we compute the decision metrics for the remaining codeword using (60).

The decision metric values of a type-II node can be computed similar to an SPC node as in (71). In particular, $(\mathbf{x}_{\text{R-4}}, \mathbf{x}_{\text{R-3}}, \mathbf{x}_{\text{R-2}}, \mathbf{x}_{\text{R-1}})$ form a $(4, 3)$ SPC node with the corresponding LLR vector $\boldsymbol{\eta} = \{\eta_0, \eta_1, \eta_2, \eta_3\}$. Let η_p denote the minimum value of $\boldsymbol{\eta}$, we can compute the decision metric corresponding to a codeword $\mathbf{x}$ which disagrees with the hard decision on $\boldsymbol{\eta}$, $H(\boldsymbol{\eta})$, at locations in $\mathbf{f}_{\mathbf{x}}$ as

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |\eta_j| - \gamma_{\text{ML}} |\eta_p|. \tag{76}$$

Remark 3: It should be noted that we do not need to compute the decision metric values using the aforementioned equations for type-I and type-II nodes. Rather, we compute the PMs for all the codewords and based on (60)*, subtract the PM of the ML codeword from that of the remaining codewords to compute the decision metrics.*

H. Type-III Node

The even and odd indexed bits of a type-III node codeword constitute two separate SPC codes. Thus, it can be optimally decoded using two Wagner decoders [6]. In particular, we set $\mathbf{x}_i^{\text{ML}} = H(\mathbf{y}_i)$ for $0 \leq i < R$. Next, we find the locations of the least reliable even-indexed and odd-indexed LLR values as $e = \underset{0 \leq i \leq R/2-1}{\arg\min} |y_{2i}|$ and $o = \underset{0 \leq i \leq R/2-1}{\arg\min} |y_{2i+1}|$ for $0 \leq i < R/2$ respectively. Then we check the parity condition for the SPC codes as

$$(\gamma_e, \gamma_o) = \left(\bigoplus_{i=0}^{R/2-1} x_{2i}^{ML}, \bigoplus_{i=0}^{R/2-1} x_{2i+1}^{ML} \right), \tag{77}$$

and set $x_{2e}^{\text{ML}} = x_{2e}^{\text{ML}} \oplus \gamma_e$ and $x_{2o+1}^{\text{ML}} = x_{2o+1}^{\text{ML}} \oplus \gamma_o$. The contribution of the ML codeword to the PM is given by

$$\text{PM}_{\text{ML}} = \Delta + \gamma_e |\mathbf{y}_{2e}| + \gamma_o |\mathbf{y}_{2o+1}|, \tag{78}$$

where $\Delta = \sum_{k=0}^{R-1} \ln\left(1 + e^{-|\mathbf{y}_k|}\right)$.

Next, we generate different codewords by flipping bits relative to the ML codeword and compute their corresponding contribution to the PM as discussed in Section III-C. It can be shown that the contribution of a type-III codeword to the PM is given by

$$\text{PM}_{\mathbf{x}} = \Delta + \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j|.$$

Here, $\mathbf{f}_{\mathbf{x}}$ is the set of bit indexes of the codeword $\mathbf{x}$ that differ from $H(\mathbf{y})$. Consequently, the decision metrics for the type-III node are

$$\lambda_{\mathbf{x}} = \sum_{j \in \mathbf{f}_{\mathbf{x}}} |y_j| - \gamma_e |\mathbf{y}_{2e}| - \gamma_o |\mathbf{y}_{2o+1}|. \tag{79}$$

I. Remaining Nodes

The fast SCF decoders for the remaining two special nodes can be similarly implemented. For the type-V node, the proposed decoder first generates 16 type-V codewords and computes their corresponding PMs. Next, it selects the codeword with the least PM as the ML codeword and computes the decision metrics for the remaining codewords using (60).

For the type-IV node, the proposed decoder works similarly to that of the proposed Type-III node. That is, it first computes the ML codeword and the corresponding PM. Next, it generates other codewords and their corresponding PMs (or equivalently the decision metrics) by flipping the bits relative to the ML codeword as detailed in Section III-D.

V. Decoding Latency

We now compare the decoding latencies of the proposed fast decoders with that of the existing ones. In particular, we compute the required number of time steps to decode different nodes with the existing decoders and our proposed ones, under the following assumptions. First, we assume there is no resource limitation so that all the parallelizable instructions are performed in one clock cycle [11], [12]. Second, addition/subtraction of real numbers and check-node operation consume one time step. Third, hard decision on LLRs and bit operations are carried out instantaneously [4], [6], [11], [12], [17]. Fourth, Wagner decoding can be performed in a single time step. Last, the decoder duplicates all L paths, sorts the corresponding $2L$ PMs and selects the smallest L ones during a single clock cycle [11], [12].

TABLE I
REQUIRED NUMBER OF TIME STEPS TO DECODE DIFFERENT NODES OF SIZE R

	Proposed fast SCL	FSCL [12]	Proposed fast SCF*	FSCF [16]
Rate-0	1	1	0	0
REP	2	2	2	2
Rate-1	$\min(L-1,R)+1$	$\min(L-1,R)+1$	$\min(T_{\max},R)$	2
SPC	$\min(L,R)$	$\min(L,R)$	$\min(T_{\max}+1,R)$	2
Type-I	3	$t+\min(L-1,2)$	2	2
Type-II	4	$t-2+\min(L,4)$	2	t
Type-III	$\min(L,R-1)$	$3t-4+\sum_{i=1}^{t-1}\min\{L-1,2^i\}$	$\min(T_{\max}+1,R-1)$	$3t-4$
Type-IV	$\min(L,R-3)$	$3t-4+\sum_{i=2}^{t-1}\min\{L-1,2^i\}$	$\min(T_{\max}+1,R-3)$	$3t-4$
Type-V	5	$t+1+\min(L,4)$	2	$t+3$

* The required number of time steps are computed for the first trial of the proposed fast SCF and FSCF decoders.

A. Fast SCL Decoder

In this section, we compare the decoding latency of our proposed fast SCL decoders with that of the existing ones [12] (referred to as FSCL decoders hereafter). Note that [12] presented fast decoders for only rate-0, REP, SPC and rate-1 nodes, and the remaining special nodes are decoded using these nodes. In the following, we first present the decoding latency of the rate-0, REP, SPC, and rate-1 node as these calculations will be used in computing the decoding latencies of the FSCL decoders for the remaining nodes.

The FSCL decoder takes 1 time step to decode a rate-0 node as computation of the PM for active paths consume a single time step. The FSCL decoder for a REP node duplicates each path and computes their corresponding PMs in a single time step. In the next time step, it sorts the PMs and selects the best L paths. As such, the decoding latency of the FSCL REP node decoder is 2 time steps. The rate-1 node FSCL decoder computes the PM of the ML codeword in 1 time step and needs $\min\{L-1,R\}$ time steps to do the path and PM updates of the remaining codewords. As such, it consumes $1+\min\{L-1,R\}$ time steps to decode a rate-1 node. Similarly, the FSCL decoder for an SPC node takes 1 time step to find the ML codeword and its corresponding PM and $\min\{L-1,R-1\}$ time steps for the path and PM updates of the remaining codewords. Thus, the decoding latency of the FSCL SPC-node decoder is $1+\min\{L-1,R-1\}$ time steps.

For type-I, type-II and type-V nodes, our proposed decoders consume one time step to compute the PMs corresponding to all codewords. Following the first and last assumptions in the beginning of this section, our decoders require 2, 3 and 4 time steps to sort $4L$, $8L$ and $16L$ PMs, respectively. As such, the decoding latencies of the type-I, type-II and type-V decoders are 3, 4 and 5 time steps, respectively.

On the other hand, if the FSCL decoders are used to decode type-I, type-II and type-V nodes then the decoding tree of these nodes correspond to the decoding tree of Fig. 2a.

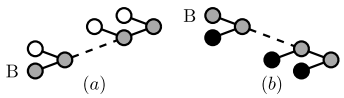

Fig. 2. Decoding trees of the proposed nodes in [6] correspond to either decoding tree (a) or decoding tree (b).

In particular, the left child of each node is a rate-0 node, whereas the node B is a rate-1 node of size 2 for the type-I node, an SPC node of size 4 for the type-II and a REP-SPC node of size 8 for the type-V node [6].

In general, the LLR computation for each child takes one time step. However, when the left children are rate-0 nodes, the LLRs of both children can be calculated simultaneously in one time step. That results in a decoding latency of $\log_2(R/R_B)$ time steps to traverse to the node B, where R_B is the size of node B. Using aforementioned decoding delay expressions, node B can be decoded in $\min\{L-1,2\}+1$, $\min\{L,4\}$, and $\min\{L,4\}+4$ time steps for type-I, type-II and type-V nodes, respectively. Using these calculations, the decoding latencies of these nodes can be computed and are given in Table I.

The proposed decoder of a type-III node first computes the ML codeword and its corresponding PM in one time step. Afterwards, it generates $2L$ codewords and select the best L ones for $\min\{L-1,R-2\}$ bits, which results in a decoding delay of $1+\min\{L-1,R-2\}$ time steps. Likewise, the decoding latency of the proposed type-IV decoder can be shown to be $1+\min\{L-1,R-4\}$ time steps.

With the FSCL decoders, the decoding tree of a type-III or type-IV node corresponds to the decoding tree of Fig. 2b. In particular, the right child of each node is a rate-1 node. The node B is a rate-0 node of size 2, and a REP node of size 4 for type-III and type-IV nodes, respectively. The decoding delays of $2\log_2(R/R_B)-1$ and $2\log_2(R/R_B)$ time steps are required to traverse to the level of node B in

type-III and type-IV node, respectively. Using the decoding delay expression of rate-0, REP and rate-1 nodes, the decoding latency of the fast SCL decoder can be computed and are given in Table I.

Observe that the decoding latencies of the proposed fast SCL decoders are less than that of the FSCL decoders. The latency improvement is more pronounced for type-III and type-IV nodes. For example, the proposed decoder will require 4 time steps to decode a type-III node, whereas the FSCL decoder will consume a total of 22 time steps to decode the node when $L = 4$ and $R = 2^t = 2^5$.

Note that the aforementioned decoding delay calculations are valid when $L > 1$ as no PM computations and path duplications are required for $L = 1$.

B. Fast SCF Decoder

The first trial of an SCF decoder, in addition to decoding the codeword, computes decision metrics and maintains a list of bit-flip positions. The remaining trials, on the other hand, do not compute any decision metric. Therefore, the decoding latency of the first trial differs from that of the remaining trials. In the following, we provide the decoding latencies of our proposed fast SCF and the existing fast SCF (FSCF) decoders [16]. Since the decoding-latency calculations are very similar to those presented in Section V-A, we briefly mention the results below.

The decoding latency of the initial trials for the FSCF and proposed SCF decoders are tabulated in Table I.

The proposed decoders for the rate-1 and SPC nodes require more time steps than the FSCF decoders. This is because the FSCF decoders for both nodes flip only single coded bit, whereas our proposed decoders flip multiple coded bits. Although the FSCF decoders consume slightly fewer time steps, they incur a significant error-correction performance loss as evident from Fig. 4.

For the remaining decoding trials, we assume that bit flipping can be carried out instantaneously. Under this assumption, the decoding latency of both the FSCF and our proposed decoders are 1, 0, 1, and 2 time steps for the REP, rate-1, SPC and type-I nodes, respectively.

For the remaining nodes, our proposed decoders require fewer time steps than the FSCF decoders. In particular, the FSCF decoders consume $t-1$, $2t-2$, $2t-3$, and $t+1$ time steps for decoding type-II, type III, type-IV and type-V nodes, respectively. The proposed decoders, on the other hand, require only 1 or 2 time steps to decode these nodes.

VI. Simulation Results

In this section, we compare the BER and block-error-rate (BLER) performances of the proposed fast SCL and fast SCF decoders with that of the existing decoders. We have used a 16-bit CRC defined by the generator polynomial $x^{16}+x^{15}+x^{12}+x^7+x^6+x^4+x^3+1$ (0xC86C), unless otherwise stated. Systematic polar codes [19] with random BPSK-modulated codewords were transmitted through AWGN channel. Furthermore, simple Gaussian approximation method [22] with a design E_b/N_0 of 4.75 dB was used to design the polar codes. We have included the performance of the SC and the oracle-assisted SC (SC-Oracle)[4] decoders in our simulation results to provide a better insight into the performance of different decoders.

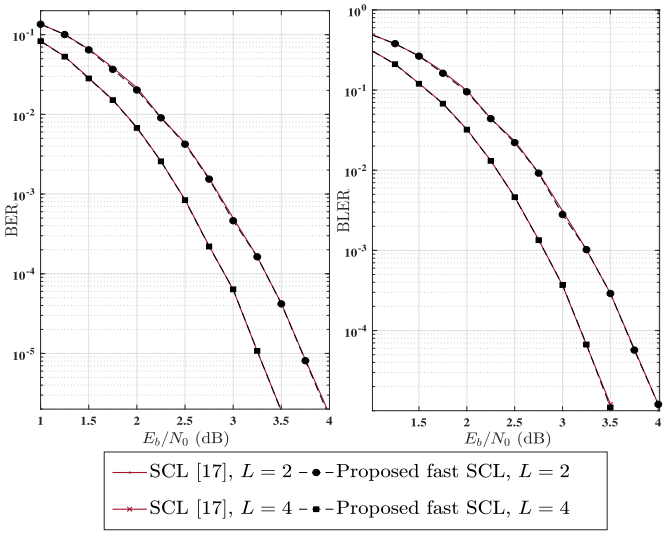

Fig. 3. BER and BLER performances of SCL and proposed fast SCL decoders for a $P(1024, 256)$ polar code.

TABLE II
NUMBER OF SPECIAL NODES FOR DIFFERENT POLAR CODES

	Node Length	Rate-0	REP	Rate-1	SPC	Type-I	Type-II	Type-III	Type-IV	Type-V
$P(1024, 256)$	8	0	9	0	4	1	0	1	1	10
	16	1	6	0	2	1	0	1	0	0
	32	1	5	0	1	0	0	0	1	0
	64	1	2	0	1	0	0	0	0	0
	128	1	0	0	0	0	0	0	0	0
$P(512, 256)$	8	0	4	1	3	1	0	0	1	6
	16	0	2	0	2	0	1	0	1	0
	32	0	2	1	1	0	0	1	0	0
	64	0	1	1	0	0	0	0	0	0
$P(128, 96)$	8	0	0	0	2	0	0	0	0	2
	16	0	0	1	0	0	0	0	0	1
	64	0	0	0	0	0	0	1	0	0

Fig. 3 compares the BER and BLER performances of the proposed fast list decoder with the LLR-based SCL decoder [17]. Note that the proposed decoder, while achieving the performance of the SCL decoder, has a decoding latency of almost 16% and 23% less than that of the FSCL decoder [12] for $L = 2$ and 4, respectively (see Table III). In particular, considering the special nodes in the simulated $P(1024, 256)$ with 16-bit CRC provided in Table II, the proposed fast SCL decoder consumes 226 and 250 time steps, whereas the FSCL decoder requires 269 and 323 time steps to complete the decoding when $L = 2$ and 4, respectively.

Fig. 4 compares the BER and BLER performances of the proposed SCF decoder with those of the SCF [15] and

[4]We employ an oracle-assisted SC decoder which is able to correct the first erroneous bit at the decision level.

TABLE III

DECODING LATENCY COMPARISON OF DIFFERENT SCL DECODERS

	Required number of time steps for proposed fast SCL		Saving with respect to FSCL [12]	
	$L = 2$	$L = 4$	$L = 2$	$L = 4$
$P(1024, 256)$	226	250	16%	23%
$P(512, 256)$	135	159	19%	25%

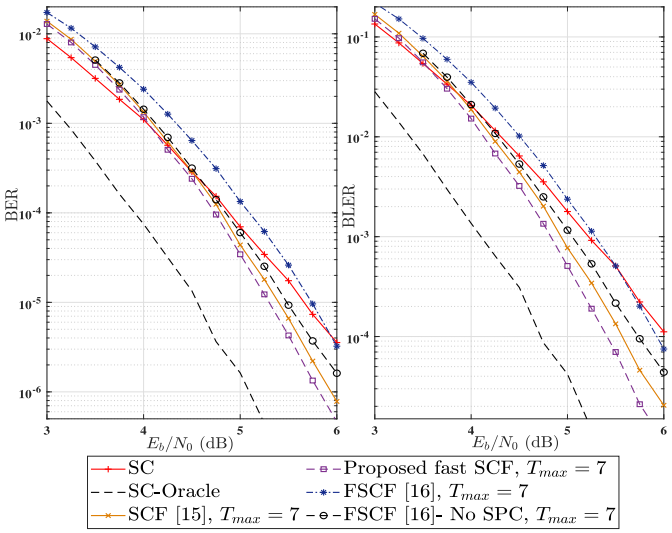

Fig. 4. BER and BLER performances of SCF, proposed fast SCF and FSCF decoders for a $P(128, 96)$ polar code and $T_{\max} = 7$.

TABLE IV

DECODING LATENCY COMPARISON OF DIFFERENT SCF DECODERS IN THE FIRST TRIAL

	Required number of time steps		Saving with respect to SCF [15]
	FSCF [16]	Proposed fast SCF	
$P(128, 96)$, $T_{\max} = 7$	51	49	81%
$P(512, 256)$, $T_{\max} = 15$	158	234	77%

FSCF [16] decoders for a $P(128, 96)$ with 8-bit CRC (0xEA). Due to the possibility of multiple bits flipping in our proposed fast SCF decoder, it slightly outperforms the SCF decoder. Furthermore, our proposed decoder significantly outperforms the FSCF decoder, whose performance degradation is mainly due to the decision metrics used in the decoding of SPC node. Note that multiple SPC nodes exist in the FSCF decoding tree of $P(128, 96)$ with 8-bit CRC. The FSCF decoder performance without SPC nodes is presented in Fig 4. It can be seen that the performance of the FSCF decoder is significantly improved without SCP nodes. The required number of time steps for the FSCF and proposed fast SCF decoders in their first trials, along with the reduction in the decoding latency using our proposed fast decoder instead of the SCF [15] decoder are provided in Table IV.

The BER and BLER performances of different SCF decoders are depicted in Fig 5. Note that the performance of our proposed fast SCF decoder is identical to that of the SCF decoder with 77% fewer required number of time steps. More specifically, our proposed fast SCF decoder consumes 234 time steps in its initial trial. Whereas, the SCF decoder requires

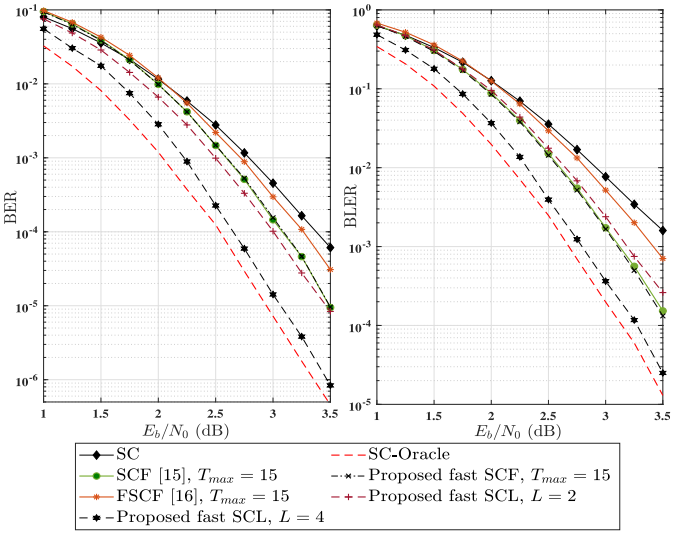

Fig. 5. BER and BLER performances of proposed fast SCL and different SCF decoders for a $P(512, 256)$ polar code.

1022 time steps. Here, we have made the assumption that finding bit flip positions is done instantaneously. Thus, the SCF decoder in each trial consumes the same number of time steps as that of the SC decoder. On the other hand, the FSCF decoder consumes 158 time steps in its first trial, however, it fails to achieve the performance of the SCF decoder. Furthermore, in the next trials, our proposed decoder requires 90 time steps while, the FSCF decoder takes 116 time steps.

Fig. 5 also illustrates the BER and BLER performances of the proposed fast SCL decoder. The required number of time steps of the proposed fast SCL decoder is 135 and 159 respectively, for $L = 2$ and 4 which is 19% and 25% less than that of the FSCL decoder [12]. Moreover, we can compare the performance of the SCF and the SCL decoders in Fig.5. It is observed that the performance of the proposed fast SCF decoder with $T_{\max} = 15$ is identical to that of the proposed fast SCL decoder with $L = 2$. However, in higher list sizes, such as $L = 4$, the proposed fast SCL decoder outperforms the proposed fast SCF decoder. Although, the proposed SCL decoder consumes fewer time steps than the proposed SCF decoder, its memory complexity is two and four times that of the proposed SCF decoder when $L = 2$ and 4, respectively.

VII. CONCLUSION

We presented fast list decoders for five recently-identified nodes, i.e., type-I, type-II, type-III, type-IV and type-V nodes. The proposed decoders, while achieving the same bit-error-rate performance, reduce the decoding latency of the list decoders significantly. We also proposed fast SC flip decoders. To that end, we first proposed a new decision metric which can be computed without the decoding tree traversal. Using the proposed decision metric, we introduced fast parallel SC flip decoders for many special nodes of the polar-code decoding tree. The proposed fast flip decoders achieve significant decoding speed improvement and, unlike the existing fast flip decoders, show the same error-correction performance as of the conventional SC flip decoder.

REFERENCES

[1] "Performance study of polar code candidates," document R1-1703538, TSG-RANWG1 #88, 3GPP, Ericsson, Athens, Greece, Feb. 2017.

[2] 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, Jul. 2009.

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

[4] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1378–1380, Dec. 2011.

[5] 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, May 2014.

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

[7] G. Sarkis, I. Tal, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Flexible and low-complexity encoding and decoding of systematic polar codes," *IEEE Trans. Commun.*, vol. 64, no. 7, pp. 2732–2745, Jul. 2016.

[8] K. Niu, K. Chen, J. Lin, and Q. T. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," *IEEE Commun. Mag.*, vol. 52, no. 7, pp. 192–203, Jul. 2014.

[9] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 Gbit/s unrolled hardware polar decoder," *Electron. Lett.*, vol. 51, no. 10, pp. 762–763, May 2015.

[10] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast list decoders for polar codes," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 2, pp. 318–328, Feb. 2016.

[11] S. A. Hashemi, C. Condo, and W. J. Gross, "A fast polar code list decoder architecture based on sphere decoding," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 12, pp. 2368–2380, Dec. 2016.

[12] 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, Nov. 2017.

[13] M. Hanif, M. H. Ardakani, and M. Ardakani, "Fast list decoding of polar codes: Decoders for additional nodes," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2018, pp. 37–42.

[14] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Multi-mode unrolled architectures for polar decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 9, pp. 1443–1453, Sep. 2016.

[15] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg, "A low-complexity improved successive cancellation decoder for polar codes," in *Proc. 48th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2014, pp. 2116–2120.

[16] P. Giard and A. Burg, "Fast-SSC-flip decoding of polar codes," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2018, pp. 73–77.

[17] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Trans. Signal Process.*, vol. 63, no. 19, pp. 5165–5179, Oct. 2015. [Online]. Available: http://arxiv.org/abs/1401.3753

[18] R. Silverman and M. Balser, "Coding for constant-data-rate systems," *Trans. IRE Prof. Group Inf. Theory*, vol. 4, no. 4, pp. 50–63, Sep. 1954.

[19] E. Arıkan, "Systematic polar coding," *IEEE Commun. Lett.*, vol. 15, no. 8, pp. 860–862, Aug. 2011.

[20] C. Condo, F. Ercan, and W. Gross, "Improved successive cancellation flip decoding of polar codes based on error distribution," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2018, pp. 19–24.

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

[22] P. Trifonov, "Randomized chained polar subcodes," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2018, pp. 25–30.

Maryam Haghighi Ardakani received the B.Sc. degree in electrical engineering from the University of Isfahan, Isfahan, Iran, in 2013, and the M.S. degree in electrical engineering from Özyeğin University, Istanbul, Turkey, in 2016. She is currently pursuing the Ph.D. degree in electrical engineering with the University of Alberta, Canada. Her research interests include channel coding, optical communications, and cooperative communications.

Muhammad Hanif received the Ph.D. degree in electrical engineering from the University of Victoria, Victoria, BC, Canada, in 2016. He is currently a Post-Doctoral Fellow with the University of Alberta, Edmonton, AB, Canada, where he is working on error-correcting codes for the 5G wireless communication systems. His research interests are in the design and performance analysis of wireless communication systems, digital signal processing, and information theory with a focus on the next-generation wireless communication systems.

Masoud Ardakani (M'04–SM'09) received the B.Sc. degree from the Isfahan University of Technology in 1994, the M.Sc. degree from Tehran University in 1997, and the Ph.D. degree from the University of Toronto, Canada, in 2004, all in electrical engineering. He was a Post-Doctoral fellow with the University of Toronto from 2004 to 2005. He is currently a Professor of electrical and computer engineering with the University of Alberta, Canada. His research interests are in the general area of digital communications. He has served as an Associate Editor for the IEEE WIRELESS COMMUNICATIONS and a Senior Editor for the IEEE COMMUNICATIONS LETTERS. He serves as an Associate Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS.

Chintha Tellambura (F'11) received the B.Sc. degree in electronics and telecommunications from the University of Moratuwa, Sri Lanka, the M.Sc. degree in electronics from the King's College, University of London, and the Ph.D. degree in electrical engineering from the University of Victoria, Canada.

He was with Monash University, Australia, from 1997 to 2002. Since 2002, he has been with the Department of Electrical and Computer Engineering, University of Alberta, where he is currently a Full Professor. He has authored or co-authored over 560 journal and conference papers, with an h-index of 71 (Google Scholar). He has supervised or co-supervised 66 M.Sc., Ph.D., and P.D.F. trainees. His current research interests include cognitive radio, heterogeneous cellular networks, fifth-generation wireless networks, and machine learning algorithms. He was a fellow of The Canadian Academy of Engineering in 2017. He received the Best Paper Awards from the IEEE International Conference on Communications (ICC) in 2012 and 2017. He is the Winner of the prestigious McCalla Professorship and the Killam Annual Professorship at the University of Alberta. He has served as an Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS from 1999 to 2012 and the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS from 2001 to 2007. He was an Area Editor of *Wireless Communications Systems and Theory* from 2007 to 2012.