Efficient Partial Rewind of Successive Cancellation-Based Decoders for Polar Codes

Mohammad Rowshan®, Member, IEEE, and Emanuele Viterbo®, Fellow, IEEE

Abstract—The successive cancellation (SC) process in which symbols are decoded sequentially by processing some intermediate information is an essential component of various decoding algorithms used for polar codes and their variants. In some decoding schemes, we may need to redo this process from some specific symbol or from the first symbol. This operation is called rewinding. Rewinding the SC process seems trivial if we have access to all intermediate log-likelihood ratios (LLRs) and partial sums. However, as the block length increases, retaining all of the intermediate information becomes inefficient and impractical. Rewinding the SC process in a memory-efficient way is a problem that we address in this paper. As we store a fraction of all the intermediate information in the memory-efficient scheme, we may not be able to rewind the SC process to the target symbol index. The reason is that some of the stored intermediate information needed to decode the target symbol may have been overwritten. To recompute the lost information, we may need to rewind the process further. Before proposing the formal scheme for the rewinding process, we explore the known properties of the SC process based on the binary representation of the bit indices. Then, we introduce a new operator used for grouping the bit indices. This special grouping helps us in finding the closest bit index to the target index for rewinding. We also analytically prove that this approach gives access to the untouched intermediate information stored in the memory which is essential in resuming the SC process. Finally, we adapt the proposed approach to multiple rewinds and apply it to SC-flip decoding and shifted-pruning-based list decoding. The numerical evaluation of the proposed solution shows a significant reduction of ≥ 50% in the complexity of the additional decoding attempts at medium and high SNR regimes for SC-flip decoding and less for shifted-pruning based list decoding.

Index Terms—Polar codes, successive cancellation, list decoding, re-decoding, bit-flipping, shifted-pruning, Fano algorithm, rewind, complexity.

I. INTRODUCTION

POLAR codes [1] are the first class of constructive channel codes that was proven to achieve the symmetric (Shannon) capacity of a binary-input discrete memoryless channel (B-DMC) using a low-complexity successive cancellation (SC) decoder. Nevertheless, the error correction performance of polar codes under SC decoding is not competitive. To address this issue, the successive cancellation list (SCL) decoding was proposed in [2] and [3] which yields an error correction performance comparable to maximum-likelihood (ML) decoding at high SNR by enhancing the distance spectrum. Further improvement was obtained by concatenating cyclic redundancy check (CRC) bits to polar codes. Decoding polar codes and their variants requires passing the channel log-likelihood ratios (LLRs) through a factor graph shown in Fig. 1. The evolved information at the output of the factor graph is used to make a hard decision or to calculate a metric in the SC-based decoders. The evolved LLR, a.k.a decision LLR, is obtained for each bit-channel successively. To calculate each decision LLR, we need to access the intermediate information on the factor graph. There are two ways to access them: 1) We can store all \( N \cdot \log_2 N \) intermediate values, including LLRs and partial sums on the factor graph. This approach is acceptable for short codes under SC decoder or Fano decoder. However, as the code gets longer, in particular under list decoding or stack decoding, this approach will be expensive in terms of memory requirements. 2) We can store a portion of the intermediate values. It was observed in [4] that for calculating every decision LLR, we need at most \( N - 1 \) intermediate LLRs (excluding channel LLRs) and \( N - 1 \) partial sums at any decoding step.

Some decoding schemes rely on additional decoding attempts when the decoding process fails in the first attempt. These schemes are as follows: 1) SC-flip decoding: In this scheme, when SC decoding fails, the decoding is repeated from scratch, while in the additional attempts, the value of a single or multiple bits is flipped throughout the SC decoding process to correct the error caused by the channel noise and avoid propagation of this error [6]. 2) Shifted-pruning-based list decoding: In this scheme [7], [8], [9], when SC list decoding fails, additional decoding attempts may correct the error given we shift the path pruning window at the position where the correct path was pruned from the list in the first decoding attempt. Note that a special case of this scheme is also called SCL flip scheme, bit-flipping for SCL, or by other names. 3) Fano decoding: In this scheme [10], [11], [12], the decoder may have a back-tracking or backward movement to explore the other paths on the decoding tree. Unlike the first two schemes where the decoding of a codeword is completed, and then the additional decoding is repeated from the first bit, in the Fano algorithm, the backward movement occurs frequently somewhere between the first bit and the last bit.
It might be better to use \( N \log N \) memory for intermediate information in Fano decoding of very short codes rather than \( N - 1 \) memory elements. This way, the complexity reduces significantly at the cost of a larger memory requirement.

In [10] and [11], we proposed a sophisticated algorithm to do the partial rewinding in Fano decoding. In this work, we propose a simple analytically-proved approach and a general mathematical framework to efficiently rewind the SC algorithm to the position that we need to flip the value of a bit in the SC-flip decoding or to shift the pruning window in the shifted-pruning scheme, a.k.a SCL-flip decoding, bit-flipping for SCL decoding, and SCL path flip decoding. This approach is designed based on the scheduling properties of the SC process and an operator that we introduce in this paper.

The contributions of this work are as follows:

- We introduce an operator called find last zero, flz that groups all the indices in \([0, N - 1]\) into disjoint sets and helps in finding the closest position to the position that the decoder should move back to, called target position. As the name suggests, this operator uses the binary expansion of the target position. Note that the closeness of the update position to the target position contributes to the significant reduction of the time and computational complexity of the underlying decoding scheme while it does retain the error correction performance relative to the full rewind, i.e., restarting the decoding from the first bit.

- We develop some general mathematical framework to show that the suggested update position, i.e., the closest position to the target position, does not require any additional update of the intermediate LLRs or partial sums. Hence, the rewind process is as simple as restarting the decoding from the suggested position and moving forward without modifying the decoder or calculating any intermediate information beforehand.

- To handle multiple iterations, the target position in each iteration is suggested based on the membership of the target position in the last iteration in the disjoint sets proposed in this work.

- Furthermore, we provide considerations for implementing the rewind process in the SC list decoding which is slightly different from the SC decoding.

**Paper Outline:** The rest of the paper is organized as follows. Section II introduces the notation for the polar codes and describes the intermediate information of the SC process. Section III reviews the details of the updating schedule for intermediate information based on the binary representation of the bit indices. In Section IV, the properties of the SC process are explored by introducing an operator and a special grouping scheme. Section V proposes a simple approach for single and multiple rewinds of the SC process. In Section VI, we evaluate the complexity reduction of the proposed approach by applying it to the SC-flip and shifted-pruning schemes. Finally, Section VII makes concluding remarks.

**II. PRELIMINARIES**

A polar code of length \( N = 2^n \) with \( K \) information bits is constructed by choosing \( K \) most reliable bit-channels in the polarized vector channel for transmitting the information bits and the optional \( r \) CRC or parity bits, hence \( K + r \) information bits in total. The indices of these bit-channels are collected in set \( A \). The rest of the \( N - K \) bit-channels are used for transmitting known values, by default 0. Polar codes are encoded by \( x_0^{N-1} = u_0^{N-1}G_N \) where \( u_0^{N-1} = (u_0, \ldots, u_{N-1}) \) is the input vector, and \( G_N = B_NG_2^{\otimes n} \), where \( G_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \). \( B_N \) is an \( N \times N \) bit-reversal permutation matrix, and \((\cdot)^{\otimes n}\) denotes the \( n\)-th Kronecker power [1]. Let \( y_0^{N-1} = (y_0, \ldots, y_{N-1}) \) denote the output vector of a noisy channel, and \( \lambda = (\lambda_0, \ldots, \lambda_{N-1}) \) indicates the log-likelihood ratios (LLRs) vector. The channel LLRs are computed based on the received signals from the physical channel, \( y_0^{N-1} \).

We also have intermediate LLRs as shown in Fig. 1. The intermediate LLRs are computed based on the type of node in the factor graph. The \( f \) and \( g \)-nodes are shown by circles and rectangles, respectively, in the factor graph. The output of these nodes can be computed from right to left by

\[
\begin{align*}
 f(\lambda_a, \lambda_b) &= \text{sgn}(\lambda_a) \cdot \text{sgn}(\lambda_b) \cdot \min(|\lambda_a|, |\lambda_b|) \\
 g(\lambda_a, \lambda_b, \hat{\beta}) &= (-1)^{\hat{\beta}}\lambda_a + \lambda_b
\end{align*}
\]

where \( \lambda_a \) and \( \lambda_b \) are the input LLRs to a node and \( \hat{\beta} \) is the partial sum of previously decided bits to feed the estimated bits \( \hat{u} \) backward into the factor graph.

In SC decoding, the non-frozen bits are estimated successively based on the evolved LLRs via a one-time pass through the factor graph. When decoding the \( i \)-th bit, if \( i \not\in A \), then \( \hat{u}_i = 0 \) since \( u_i \) is a frozen bit. Otherwise, bit \( u_i \) is decided by maximum likelihood (ML) rule \( h(\lambda_0) = (1 - \text{sgn}(\lambda_0))/2 \). Unlike SC decoding which makes a final decision for the \( i \)-th bit, SC list decoding considers both possible values \( u_i = 0 \) and \( u_i = 1 \). In SC list decoding, the \( L \) most reliable paths are preserved at each decoding step to limit the growth of the number of paths. The solution for decoding is decided at the last bit based on path likelihood or the cyclic redundancy check (CRC) approach. The cyclic redundancy check (CRC) is also used in the re-decoding schemes such as SC-flip and the shifted-pruning-based list decoding.

**A. Updating the Intermediate LLRs**

The factor graph shown in Fig. 1 has \( N \log_2 N \) nodes however, as it was shown in [4], it is sufficient to update/access at most \( N - 1 \) intermediate LLRs out of \( N \log_2 N \) LLRs for decoding any bit \( u_i \), \( 0 \leq i \leq N - 1 \). Fig. 1 illustrates the LLRs associated with decoding bit \( u_3 \) denoted by \( \lambda_0, \lambda_1, \ldots, \lambda_{14} \). As can be seen, there are \( 2^s \) LLRs in each stage \( s \) for \( s = 0, \ldots, n \). Hence, according to geometric series, we need a memory space of

\[
\sum_{s=0}^{n} 2^s = 2^{n+1} - 1 = 2N - 1 \tag{3}
\]

elements in total. Suppose \( u_i \) is the bit to be decoded and \( \text{bin}(i) = i_{n-1} \ldots i_0 \) is the binary representation of index \( i \) where the least significant bit is indexed 0 (the rightmost bit) and most significant bit is indexed \( n - 1 \). The stages are updated from right (where \( s = n \)) to left (where \( s = 0 \)). The first stage to be updated is obtained by finding the first one, \( \text{ffo} \), or the position of the least significant bit set to one as

\[
\eta(i) = \text{ffo}(i_{n-1} \ldots i_0) = \begin{cases} 
\min(t) & i > 0 \\
 n - 1 & i = 0 
\end{cases} \tag{4}
\]
A memory-efficient way of updating the LLRs has been used in non-hardware-based works such as in the well-known algorithm proposed for list decoding in [3] and even in the block sequential decoding in [5] where a block code is decomposed into its constituent codes. Note that the full rewind in the local decoder of any constituent codes is as straightforward as the full rewind of the decoder of the whole block code and we do not need to worry about the possibility of corruption of shared LLRs among the constituent codes. One can readily observe this fact in [13].

B. Partial Sums

The Partial sums are the other set of intermediate information needed for the SC process. Unlike the LLRs, partial sums are updated in the opposite direction on the factor graph in Fig. 2, that is, from left to right. It turns out that we need the same memory space for the partial sums as well, i.e., at most $N - 1$ memory elements. It was observed in [4] that we need to store $2^s$ bits corresponding to $2^s$ nodes of type $g$ at stage $s$, which are waiting to be summed with the next decoded bit. Here, let us define an operator that indicates the last stage to be updated. The last stage that its partial sums to be updated is obtained by finding the first zero, ffz, or the position of the least significant bit set to zero as

$$\psi(i) = \text{ffz}(i_{n-1} \cdots i_0) = \min_{i=0}^{n-1}(i). \quad (5)$$

It turns out that this is the only stage that consists of $g$-nodes in the process of updating LLRs from stage $s = \text{ffo}(\text{bin}(i))$ up to $s = 0$. Clearly, after decoding the final bit with index $N - 1$ where there is no zero in the binary representation of the index, $\text{bin}(N - 1) = 11 \ldots 1$, there is no need to update partial sums as there is no further bit to decode.

Fig. 2 shows $N - 1 = 8 - 1 = 7$ partial sums ($\beta_0$ to $\beta_6$) associated to $u_3$. The $\beta$ values in orange are updated after decoding bit $u_3$ as follows: $\beta_0 = u_3$, $\beta_5 = u_3 \oplus \beta_0$, $\beta_4 = u_3 \oplus \beta_2$, and $\beta_3 = u_3 \oplus \beta_1 \oplus \beta_2$.

There are methods proposed in [14] and [15] for hardware implementation that require slightly less memory space for updating the partial sums.

You may notice that for $i \in [0, N - 2]$, we have

$$\psi(i) = \eta(i + 1) \quad (6)$$

That is the reason why at any bit $i \in [1, N - 1]$, the stage $\eta(i)$ where its LLRs need to be updated consists of only $g$-nodes. Therefore, after decoding bit $i - 1$, the partial sums of this stage are updated to be used for the $g$-nodes at stage $\eta(i)$.

III. PROPERTIES OF THE SC PROCESS

We discover some properties of the SC algorithm that can help us to rewind the process efficiently. The goal is not storing all the $N \log_2 N$ values for LLRs and partial sums or restarting the SC process from bit 0 in the SC-based decoding when a re-decoding attempt is required. First, let us define an operator that helps us in the upcoming analysis.

**Definition 1.** The operator $\phi(j)$ finds the last zero, ffz, or the position of the most significant bit set to zero in the binary representation of $j = (j_{n-1} \cdots j_0)$ indexed in reverse order as

$$\phi(j) = \text{ffz}(j_{n-1} \cdots j_0) = \begin{cases} n - 1 - \max(t) & j < 2^n - 1, \\ n - 1 & j = 2^n - 1 \end{cases} \quad (7)$$

for every $t \in [0, n - 1]$. We denote the output of the operator $\phi(j)$ by parameter $p$.

Note that since the indexing is in the opposite direction when the most significant bit is set to zero, i.e., $j_{n-1} = 0$, then we get $p = 0$, and when the only zero bit is $j_0 = 0$ or there is no 0-value bit, then $p = n - 1$.

**Definition 2 (Set $Z_p$):** We group the bit indices $j \in [0, 1, \ldots, 2^n - 1]$ based on the identical $p = \phi(j)$ into $n$ sets denoted by set $Z_p$ with order $p = 0, 1, \ldots, n - 1$, or

$$Z_p = \{ j \in [0, 2^n - 1] : \phi(j) = p \} \quad (8)$$

**Example 1:** For $n = 3$, we can group the indices 0 to 7 into the following sets: $Z_0 = \{ 0, 1, 2, 3 \}$, $Z_1 = \{ 4, 5 \}$, and $Z_2 = \{ 6, 7 \}$.
Remark 1: The distribution of non-frozen indices in set $A$ among sets $Z_{p}$, $p \in [0, n-1]$ depends on the code rate. As the code rate reduces, fewer non-frozen indices will exist in low order $Z_{p}$, i.e., $Z_{p}$ with small $p$.

Lemma 1 (Properties of $Z_{p}$): For any $n > 0$ and $p \in [0, n-1]$, set $Z_{p}$ has the following properties:

a. The boundaries of set $Z_{p}$ are

$$ Z_{p} = \left\{ \begin{array}{ll}
2^n - 2^{n-p}, & 0 \leq p < n-1, \\
2^n - 2^{p}, & p = n-1
\end{array} \right. $$

(9)

b. The size of set $Z_{p}$ is

$$ |Z_{p}| = \begin{cases} 2^{n-p-1} & 0 \leq p < n-1, \\ 2 & p = n-1 \end{cases} $$

(10)

c. The smallest element in set $Z_{p}$ is

$$ z_{p} = \min(Z_{p}) = \min_{x=n-p} 2^{x} = 2^{n} - 2^{n-p} $$

(11)

Proof: Let us first introduce a notation for the binary representation of a positive integer with length $n$. Denote by $\{0, 1\}^{2}$ a mixed string of 0’s and 1’s, and let $\{b\}_{x}$, $b \in \{0, 1\}$ denote a uniform string of either 0 or 1, both with length $x$. In set $Z_{p}$, $p < n-1$, observe that the elements are in the form of $\{1\}^{p} + \{0\} + \{0, 1\}^{n-(p+1)}$, where the operator $\ast$ is used for concatenation and $\{1\}^{p}$ is the most significant bits.

a. The smallest element of set $Z_{p}$ in binary is

$$ \{1\}^{p} + \{0\} + \{0\}^{n-(p+1)} $$

which is equivalent to

$$ \sum_{x=n-p}^{n-1} 2^{x} = 2^{n} - 2^{n-p} $$

in decimal. Similarly, one can see that the largest element in set $Z_{p}$, $p < n-1$ is

$$ \{1\}^{p} + \{0\} + \{1\}^{n-(p+1)} $$

$$ = (\{1\}^{n})_{2} - (\{1\} + \{0\}^{n-(p+1)})_{2} $$

which is equivalent to $(2^{n-1} - 2^{n-p})$ in decimal. Note that the largest element in set $Z_{p}$, $p < n-1$ is $\{1\}^{n}_{2} - 2^{n-1}$ while the smallest element follows the relationship discussed above.

b. The given interval $[\min(Z_{p}), \max(Z_{p})]$ is part a of this lemma, we can find the size of set $Z_{p}$ by

$$ \max(Z_{p}) - \min(Z_{p}) $$

which follows from part a of this lemma that the lower bound of the values in set $Z_{p}$ in binary is $\{1\}^{p} + \{0\}^{n-p}$ which is equivalent to $2^{n-p}$ in decimal.

Example 2: For $n = 4$, we have $z_{p}$ for $p = 0, 1, \ldots, n-1$ as

$$ z_{0} = (0000)_{2}, z_{1} = (1000)_{2} = 8 $$
$$ z_{2} = (1100)_{2} = 12, \text{ and } z_{3} = (1110)_{2} = 14 $$

or based on the lower bound of $Z_{p}$ in Lemma 1 as

$$ z_{0} = 2^{n} - 2^{n-0} = 16 - 2^{4} = 0, z_{1} = 16 - 2^{3} = 8 $$
$$ z_{2} = 16 - 2^{2} = 12, \text{ and } z_{3} = 16 - 2 = 14 $$

Let us find the deepest updated stage while decoding any bit $i$ within set $Z_{p}$ in the following lemma.

Lemma 2: For any $i \in Z_{p}$, $p \in [0, n-1]$, and $z_{p} = \min(Z_{p})$ we have

$$ \max_{i \in Z_{p}}(\eta(i)) = \eta(z_{p}) $$

(12)

Proof: Let us recall the notation $\{1\}^{p} + \{0\} + \{0, 1\}^{n-(p+1)}$ for $i \in Z_{p}$, $p < n-1$ where the operator $\ast$ is used for concatenation and $\{1\}^{p}$ is the most significant bits. According to (4), the maximum value for $\eta(i)$, i.e., the largest index for the least significant bit set to one for $i \in Z_{p}$, is obtained when we have

$$ \bin(i) = \{1\}^{p} + \{0\} + \{0\}^{n-(p+1)} = \{1\}^{p} + \{0\}^{n-p} $$

which is the smallest element in set $i \in Z_{p}$, $p < n-1$, i.e.,

$$ z_{p} = \min(Z_{p}) $$

For $p = n-1$, although the notation is in the form of $\{1\}^{n-1} + \{0, 1\}^{1}$, the largest index for the least significant bit set to one is similarly obtained from $\{1\}^{n-1} + \{0\}$ which is the smallest in set $Z_{n-1}$.

Clearly, when $p = 0$, we have $\max(\eta(i)) = \eta(0) = n-1$ for any $i \in Z_{0}$.

Remark 2: From Lemma 2 we conclude that the deepest stage that the intermediate LLRs are overwritten is when decoding the smallest bit index in set $Z_{p}$. Recall that the partial sums used at stage $\eta(z_{p})$ are provided after decoding bit with index $z_{p} - 1$ according to (6).

Now consider updating intermediate information for $i$ in different sets of $Z_{p}$.

Lemma 3: For any $p, p' \in [0, n-1], p < p'$, we have

$$ \eta(z_{p}) > \eta(z_{p'}) $$

(13)

Proof: It follows directly from (4) and part c of Lemma 1. Note that $z_{p}$ is in the form of $\{1\}^{p} + \{0\}^{n-p}$. It can be observed that for the smaller $p$, the position of the least significant bit set to one has a larger index. Therefore, $\eta(z_{p})$ is larger.

Corollary 1: For any $p, p' \in [0, n-1], p < p'$, we have

$$ \psi(z_{p} - 1) > \psi(z_{p'} - 1) $$

(14)

Proof: As $\eta(z_{p}) = \psi(z_{p} - 1)$ according to (6), then based on Lemma 3, it follows that $\psi(z_{p} - 1) > \psi(z_{p'} - 1)$.

Remark 3: From Lemma 3 and Corollary 1, we conclude that intermediate LLRs and partial sums of stage $\eta(z_{p})$ are not overwritten when we are decoding any bit with index $i \in Z_{p'}$, $p < p'$. Hence, Remark 4: As per Remark 3 and the fact that updating the intermediate information is performed from stage $\eta(z_{p})$ to stage 0, reworking the SC algorithm from bit $i \in Z_{p}$ to bit $z_{p}$, $p < p'$ does not require any additional update of the intermediate LLRs or partial sums.

We will use remarks 3 and 4 in the proposed approach later.

IV. EFFICIENT PARTIAL REWIND

Since we use limited space for intermediate information, i.e., $2N-1$ memory elements instead of $N + N \log_{2} N$ as discussed in Section II-A and II-B, we have to overwrite the current values we no longer need to proceed with decoding. In the normal decoding process, the overwriting operation does not cause any data corruption. However, if we need to
move backward like in SC-flip, SCL based on shifted-pruning, or Fano decoding, we may no longer access the intermediate information as it may have been lost due to overwriting.

In this section, based on the properties of the SC process studied in Section III, a scheme is proposed such that rewinding the SC algorithm is performed efficiently by significantly fewer computations compared to restarting the algorithm.

Suppose the SC algorithm is decoding bit \( i \) and needs to rewind the SC process to bit \( j, j < i \), and \( i, j \in \mathcal{A} \). In SC-flip scheme and shifted-pruning scheme, we have \( i = 2^n - 1 \) however, in Fano decoding, \( i \in 2^n - 1, i \in \mathcal{A} \). Since the required intermediate information for decoding bit \( j \) may partially be overwritten, we may need to rewind further to a position denoted by \( j_p \). From \( j_p \), the SC algorithm proceeds with the normal decoding up to position \( j \). We shift the pruning window at this position, or we flip the bit \( u_j \) and then continue the normal SC-based decoding.

Now, the question is what the position \( j_p \) is. Let us assume \( i \in \mathcal{Z}_{p'} \) and \( j \in \mathcal{Z}_p \). Then,

\[
 j_p = \begin{cases}  
 z_p & \text{if } z_p < z_{p'} \\
 z_{p'} & \text{if } z_p = z_{p'} 
 \end{cases} 
\]  

(15)

**Example 3:** Suppose \( N = 2^5 \) and we need to rewind the SC algorithm from position \( i = 31 = (11111)_2 \) to \( j = 19 = (10011)_2 \). We know that \( i \in \mathcal{Z}_4 \) and \( j \in \mathcal{Z}_1 \). Therefore, according to (15), \( j_p = z_p = 16 = (10000)_2 \).

**Recursion for Case \( z_p = z_{p'} \):** For the case \( z_p = z_{p'} \) in (15), we may choose a position \( k, j_p < k \leq j \) for rewinding, which is more efficient. To this end, let us \( k \leftarrow j \) and \( m \leftarrow n \), then while \( \phi(k) \neq 0 \):

- first, truncate the binary representation of \( k = (k_{n-1} \ldots k_0)_{10} \) by removing the bits from position \( m = 1 - \phi(k) \) to the most significant bit (inclusive), i.e., position \( m - 1 \). Note that after truncation, we have a binary number with length \( m = m - (\phi(k) + 1) \).
- secondly, find the new set \( \mathcal{Z}_{p'} \) such that \( k \in \mathcal{Z}_{p'} \) for \( k = \sum_{t=m}^{n-1} (\phi(k) + 1) j_t \cdot 2^t - k \).
- then, \( j_p = \sum_{t=m}^{n-1} (\phi(k) + 1) j_t \cdot 2^t + z_{p'} \).

We can continue the above procedure recursively to minimize \( z_{p'} - j \). Note that in this recursion, \( k \) and \( m \) are being replaced with new values at each iteration.

**Example 4:** Suppose \( N = 2^5 \) and we need to rewind the SC algorithm from position \( i = 22 = (10110)_2 \) to \( j = 19 = (10011)_2 \). We know that \( i, j \in \mathcal{Z}_1 \) and therefore \( j_p = z_p = 16 = (10000)_2 \). We truncate \( j = (10011)_2 \) as mentioned above. We get \( k = (0112)_2 \), \( k \in \mathcal{Z}_0 \), and \( z_{p'} = 0 \). Hence, the new \( j_p \) is \( j_p = z_p + z_{p'} = 16 \) which is the same as before.

**Example 5:** Suppose \( N = 2^5 \) and we need to rewind the SC algorithm from position \( i = 22 = (10110)_2 \) to \( j = 20 = (10100)_2 \). We know that \( i, j \in \mathcal{Z}_1 \) and therefore \( j_p = z_p = 16 = (10000)_2 \). However, if we truncate \( j = (10100)_2 \) as mentioned above, we get \( t = (1002)_2 \), \( t \in \mathcal{Z}_1 \), and \( z_{p'} = 4 \). Hence, the new \( j_p \) is \( j_p = z_p + z_{p'} = 16 + 4 = 20 \).

Observe that the recursion is not used in the schemes where the rewind is performed from the last bit, for instance, in the SC-flip decoding. The reason is that bit index \( 2^n - 1 \in \mathcal{Z}_n \) and this set has only one other element which is \( 2^n - 2 = z_{n-1} \).

On the other hand, if we need to rewind the SC process to a bit index smaller than \( z_{n-1} \), the target bit index will fall into another set \( \mathcal{Z}_p \) with different \( z_p \). Hence, this may be used for Fano decoding where the case \( z_p = z_{p'} \) is possible. Note that we do not numerically evaluate this approach for Fano decoding as we do not have any other approach to compare with. We can either use this approach or simply we can store all \( N \log_2 N \) intermediate LLRs and partial sums and trade a significant complexity reduction with the memory efficiency.

Now, let us adapt the proposed approach for rewinding more than once. In the shifted-pruning scheme (and in the SC-flip scheme), we may need to repeat the rewind of the SC process up to \( T \) times. Therefore, we need to take this into our consideration. Assuming \( t \in [1, T] \) indicates the current iteration, and \( j(t) \) and \( j_p(t) \) denotes the \( j \) and \( j_p \) of iteration \( t \), then \( j_p \) of the current iteration is obtained by considering \( j_p(t-1) \) as follows:

\[
 j_p = \begin{cases}  
 j_p(t-1) & \text{if } j_p(t) > j_p(t-1) \\
 j_p(t) & \text{otherwise} 
 \end{cases} 
\]  

(16)

As (16) shows, if the destination position of the current iteration \( j(t) \) is larger than the destination position of the previous iteration, the intermediate information is not valid. The reason is that some modification (bit-flipping or shifted-pruning) occurred at position \( j(t-1) \) that affects not only the intermediate information but also the decoded data. In other words, we need to go to position \( j(t-1) \) and undo the modification then proceed with the decoding up to position \( j(t) \), and finally perform the modification of the current iteration. Note that if both \( j(t) \) and \( j(t-1) \) are in the same \( \mathcal{Z}_p \), then \( j_p(t) = j_p(t-1) \), hence there will be no difference.

Fig. 3 compares \( j \) and \( j_p \) for an example when 5 iterations are occurring. The solid arrows at the top of the axis show the bit position \( j \) where the decoder needs to restart from and the corresponding dashed arrows having a similar color as the solid arrows point to the bit position \( j_p \) where we can restart decoding instead of bit position \( 0 \). For instance, the solid arrow 1 illustrates the case when the decoder needs to flip a bit at the position after index \( N - 2^x \) for some \( x \) (and before index \( N - 2^{x-1} \leq N - 1 \)). The decoder has to start from position \( N - 2^x \) shown by dashed arrow 1. Note that in Fig. 3, the order of iterations indicated by numbers is important because the update position \( j_p(t) \) at iteration \( t \) depends on the update position \( j_p(t-1) \) of the previous iteration as discussed in (16). For instance, in the 5-th iteration (\( t = 5 \)) shown by gray arrows, since \( j_p(4) = N - 2^{n-1} \), \( j_p(5) \) cannot be \( N - 2^{x+1} \) but it should be \( j_p(5) = j_p(4) = N - 2^{n-1} \).

The distribution of complexity with respect to the range of bit indices can be described as follows: Suppose the number of operations in (1) and (2) is the measure of complexity (assuming the complexity of both are the same) and the target bit index for flipping/shifting is \( j \). According to (4), the number of operations needed for an additional iteration is

\[
\begin{cases}  
 2^n \cdot n & \text{if } j \in [0, 2^{n-1}-1], \\
 2^x \cdot (x+1) & \text{if } j \in [N-2^x, N-2^{x-1}-1] \text{ for } 1 \leq x < n 
\end{cases} 
\]  

(17)

Note that this is the minimum number of operations because according to (16), more operations may be needed depending on the target bit index in the previous iteration.
Algorithm 1 illustrates the use of rewinding in the SC-flip algorithm in line 26 where the subroutine \texttt{RewindTo} returns the update position \( z \) which is smaller or equal to the target position \( \delta[t, 1] \). As can be seen, depending on the iteration counter \( t \), either the decoder starts decoding from bit \( i = 0 \) in lines 18-23 when \( t = 0 \), or from bit \( i = z \) in lines 27-32 when \( t > 0 \). In the latter case, \( u_i \) is flipped at position \( i = \delta[t, 1] \) in line 31.

Furthermore, when rewinding the list decoder from the last bit position, \( N - 1 \), to position \( j_p \), some of the paths that existed at position \( j_p \) in the previous iteration might be eliminated in between and be replaced with other paths. This potential replacement should be addressed when we have a list of paths/candidates, such as in the shifted-pruning scheme, not in the SC-flip scheme. To simplify the problem, we can limit the positions \( j_p \) to \( j_p = 2^n - 1 \). Because all the computations of the intermediate LLRs from this position, \( 2^n - 1 \), to the last position, \( 2^n - 1 \), are performed solely based on the channel LLRs and partial sums of stage \( \psi(2^{n-1} - 1) \). Hence, we need to store the decoded data, \( u[0 : 2^{n-1} - 1] \), and the path metric of all the paths at position \( 2^{n-1} - 1 \). Partials sums can be stored as well or can be computed simply by \( u[0 : 2^{n-1} - 1] \).

\section{V. Numerical Results}

We show that in the additional decoding attempts (constrained to \( T = 10, 20 \) iterations) in the SC list decoding and SC decoding, the average complexity (in terms of required time-steps and node visits) can be significantly reduced by partial rewinding instead of full rewinding of the SC-based decoders. Here, one time-step is considered a significant time in which all the active nodes in one stage of the factor graph shown in Fig. 1 are processed. Note that taking average over all the decoding attempts including the successful attempts in the first run does not give a good insight and we cannot make a fair comparison in particular at medium and high SNR regimes. The reason is that only a small portion of the total attempts fail requiring additional attempts, e.g., less than 10 failures in \( 10^4 \) decoding attempts in the frame error rate (FER) of \( 10^{-4} \). Hence, the impact of this small portion becomes negligible on the average number of total attempts per codeword at high SNR regimes. Therefore, in the numerical results below, we analyse the average complexity of the additional iterations (when decoding fails in the first iteration) as well. Any significant reduction in the complexity of the additional iterations, for instance in terms of time-steps, is important from a practical standpoint. In practice, the signals are received constantly by the receiver in a communication system. If we spend many time-steps correcting the errors of one frame, we will require a large input buffer to store the received signals in a queue to be processed later. The need for
a large input buffer is costly in terms of energy consumption and silicon area.

Figures 4 and 5 compare the average computational complexity of shifted-pruning scheme with and without partial rewinding for two different codes. In Fig. 4, the FER and time complexity of polar code of $(512,256+12)$ constructed with DEGA (2dB) [16] and concatenated with CRC12 with polynomial $0xC06$ under SC list decoding with list size $L=8$ with shifted-pruning (SP) are shown. The FER before and after using the efficient partial rewind (PR) scheme clearly shows that the proposed efficient partial rewind scheme does not degrade the decoder’s performance as we expected. However, it reduces the average time-steps over additional iterations (when the decoding fails) by over 30% (from $2N-2=1022$ time-steps down to about 700 time-steps). The average time steps over all the iterations also reduce, but at high SNRs, it approaches 1022. The reason is that at high SNR regimes, the number of errors, FER, is low. Compared with the total number of codewords decoded successfully, just a small number of codewords failed to be decoded in the first attempt and need additional attempts/iterations.

As Fig. 5 shows, the reduction in the average time complexity for the efficient partial rewind scheme improves for polar code $P(512,128+12)$ constructed with DEGA (1dB). The average time-steps over additional attempts by about 45% (from $2N-2=1022$ time-steps down to about 570 time-steps). The reason is that at a low code rate of $R=1/4$, the positions $j$ for shifting the pruning window are mostly located in the interval $[N/2, N-1]$ where partial rewinding can be effective in reducing the complexity. Recall that if $j \in [0, N/2-1] = \mathcal{Z}_0$, then $j_p = z_p = 0$. That means a full rewind is needed. One can guess that the reduction in the complexity would be less at high code rates where the position $j$ for shifting are dominantly located in $[0, N/2-1] = \mathcal{Z}_0$ as the reliability of these bit-positions are less relative to the ones in $[N/2, N-1]$.

Similarly, we can show a significant reduction in the complexity of the additional attempts in the SC-flip decoding algorithm. Fig. 6, 7, and 8 illustrate the reduction in the node visits on average for CRC-polar codes of length $N=512$ at rates $R = 1/4, 1/2, 3/4$. The metric used in the SC-flip implementation is similar to the one in [6] as our purpose in this work is not the performance of SC-flip but to show the reduction in the complexity. Hence, a similar result can be obtained by applying the partial rewind on any variant of the SC-flip decoder. As can be seen, the FER remains unchanged by partial rewind, while the additional decoding...
Fig. 7. Comparison of FER and average node visits of \( P(512,256+12) \) under SC decoding without and with (w/) bit-flipping, and with partial rewinding (PR). ‘all’ and ‘add’ indicate averaging over all the decoding iterations and averaging only over additional iterations, respectively.

Fig. 8. Comparison of FER and average node visits of \( P(512,388+12) \) under SC decoding without and with (w/) bit-flipping, and with partial rewinding (PR). ‘all’ and ‘add’ indicate averaging over all the decoding iterations and averaging only over additional iterations, respectively.

Fig. 9 compares the time complexity at at rates \( R = 1/4, 1/2, 3/4 \). By recalling Remark 1, one can observe that at low code rates, \((N-1) - J_p\) on average decreases significantly compared with high rates, therefore, we expect to visit fewer nodes in the additional decoding attempts and consequently the time complexity reduces more than high code rates. Similar to node visits, this reduction increases at high SNR regimes as the targeted positions for bit-flipping become more accurate and their number decreases. Note that the average time complexity over additional iterations does not depend on the code rate if we don’t use partial rewinding as we start redecoding from bit 0 for any code rate. Furthermore, given the maximum number of iterations \( T \), the maximum time-steps in the worst-case scenario is simply \( T \cdot (2N - 2) \) where \( 2N - 2 \) is the total time-steps required for one full iteration. This is the case for both schemes in the worst-case scenario.

For further numerical analysis, we consider the number of operations as a measure of complexity. It is important to know that the number of operations required for decoding every bit is not constant and it varies throughout the decoding process.
depending on the activated nodes in the stages of the factor graph.

Now, we consider a fixed maximum budget of additional operations for $T$ iterations, that is, $T \cdot N \cdot \log_2 N$, in both full rewind and partial rewind schemes. This is equivalent to an increase in the maximum number of iterations under partial rewind schemes because in the partial rewind scheme, the decoder does not perform $N \cdot \log_2 N$ operations in each additional iteration. For the cases where the maximum $T$ iterations are not enough to correct the error, this effective increase in $T$ may help the decoder to correct the error. Hence, we expect to improve the error correction performance. Fig. 10 shows that by fixing the maximum number of operations in both full rewind and partial rewind schemes, the error correction performance improves by up to 0.2 dB power gain. This improvement costs a slight increase in the average number of operations (indicated by ‘add’) though it is still lower than the full rewind scheme’s. Moreover, the average number of operations in additional iterations (indicated by ‘all’) remains almost the same.

VI. CONCLUSION

When decoding fails in the first decoding attempt, a partial rewind of the SC process for additional attempts is needed in the memory-efficient SC-based decoders. In this paper, an efficient partial rewinding approach based on the properties of the SC algorithm is proposed. This approach relies on the properties of the SC process and its updating schedule. Then, this scheme is adapted to multiple iterations, and to the SC list decoding, where there exists more than one path compared with the SC decoding. The numerical results show a significant reduction in the average time and computational complexity of the additional decoding attempts while the performance remains the same. Observe that if the average complexity of the additional iterations would be large, the iterative schemes would not be practical as they require large input and output buffers to handle the constant stream of receiving data. Hence, although the average complexity at high SNR regimes approaches the complexity of full-rewind schemes, any reductions in the additional attempts would make a big difference.

REFERENCES

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," IEEE Trans. Inf. Theory, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[2] I. Tal and A. Vardy, "List decoding of polar codes," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Saint Petersburg, Russia, Aug. 2011, pp. 1–5.

[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] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, "Hardware implementation of successive-cancellation decoders for polar codes," J. Signal Process., vol. 69, no. 3, pp. 305–315, Dec. 2012.

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

[6] 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.

[7] M. Rowshan and E. Viterbo, "Improved list decoding of polar codes by shifted-pruning," in Proc. IEEE Inf. Theory Workshop (ITW), Visby, Sweden, Aug. 2019, pp. 1–5.

[8] M. Rowshan and E. Viterbo, “Shifted pruning for path recovery in list decoding of polar codes,” in Proc. IEEE 11th Annu. Comput. Commun. Workshop Conf. (CCWC), Jan. 2021, pp. 1179–1184.

[9] Y. Lv, H. Yin, and Y. Wang, “An adaptive ordered shifted-pruning list decoder for polar codes,” IEEE Access, vol. 8, pp. 225181–225190, 2020.

[10] M. Rowshan, A. Burg, and E. Viterbo, “Polarization-adjusted convolutional (PAC) codes: Sequential decoding vs list decoding,” IEEE Trans. Veh. Technol., vol. 70, no. 2, pp. 1434–1447, Feb. 2021.

[11] M. Rowshan, A. Burg, and E. Viterbo, “Complexity-efficient Fano decoding of polarization-adjusted convolutional (PAC) codes,” in Proc. Int. Symp. Inf. Theory Appl. (ISITA), Oct. 2020, pp. 200–204.

[12] P. Yuan and M. C. Coskun, “Complexity-adaptive maximum-likelihood decoding of modified GN-coset codes,” in Proc. IEEE Inf. Theory Workshop (ITW), Oct. 2021, pp. 1–6.

[13] 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.

[14] G. Berhault, C. Leroux, C. Jego, and D. Dallet, “Partial sums computation in polar codes decoding,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2015, pp. 826–829.

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

[16] P. Trifonov, “Efficient design and decoding of polar codes,” IEEE Trans. Commun., vol. 60, no. 11, pp. 3221–3227, Nov. 2012.