Stepped List Decoding for Polar Codes

Mohammad Rowshan, *Student Member, IEEE* and Emanuele Viterbo, *Fellow, IEEE*
ECSE Department, Monash University, Melbourne, VIC 3800, Australia*
Email: {mohammad.rowshan, emanuele.viterbo}@monash.edu

***Abstract*—In the successive cancellation list (SCL) decoding of polar codes, as the list size increases, the error correction performance improves. However, a large list size results in high computational complexity and large memory requirement.**

In this paper, we investigate the list decoding process by introducing a new parameter named path metric range (*PMR*) to elucidate the properties of the evolution of the path metrics (PMs) within the list throughout the decoding process. Then, we advocate that the list size can change stepwise depending on *PMR*. As a result, we propose a stepped list decoding scheme in which the error correction performance of the conventional list decoding is preserved while the path memory may reduce by 75%, the size of the internal LLR memory and partial sums memory can drop by 50%, and the computational complexity may halve. The reduction in complexity is SNR-independent and achieved without introducing any computational overhead.

***Index Terms*—Polar codes, successive cancellation list decoding, computational complexity, memory requirement.**

I. Introduction

The polar codes proposed by Arikan [1] have good properties such as capacity-achieving, and availability of efficient encoding and decoding algorithms. Nevertheless, the polar codes under successive cancellation (SC) decoding suffer from poor error correction performance for short and medium block-lengths. To address this issue, the SCL decoding and CRC-aided SCL decoding were proposed [2].

Although the CRC-aided SCL decoder provides a competitive performance, its main drawbacks are high computational complexity and large memory requirement. To reduce the size of storage and processing elements in a hardware implementation, the internal soft messages were changed from log-likelihood (LL) to LLR in [3]. However, the total memory area still accounts for 40%-45% of the total silicon area.

In another attempt, the tree/list pruning method was proposed to reduce the complexity. In this method, the path list is pruned using a threshold obtained either online or offline [4], [5]. Although this method introduces a computational overhead in list pruning procedure, it can reduce the overall computational complexity substantially. In [6], the computational complexity was reduced by dropping the frequently split paths from the list. Additionally, a counter was used to recognize the correct path and then it was switched to SC decoder for decoding the rest of the bits. Nevertheless, this method cannot provide the performance of CRC-aided decoder. Also, similar to the tree pruning method, it requires the conventional (memory-intensive) SCL decoding.

* This research work is supported by the Australian Research Council under Discovery Project ARC DP160100528.

Segmenting or partitioning based on multi-CRC schemes proposed in [7], [8] is another method in which the memory requirement reduces. In this method, every partition estimates a sub-block of the code by performing CRC-aided SCL decoding and then the estimated bits are passed to the next partition. Although this method saves the memory significantly and contributes slightly in the complexity reduction, it requires to employ several short CRCs at the cost of increase in the effective code rate, which consequently affects the error correction performance. For instance, $4 \times 8 = 32$ bits are used for CRCs in 4-partition scheme, compared with 16 bits in conventional CRC-aided SCL decoding.

This work investigates the evolution of path metrics within the list throughout the decoding process and suggests to employ a new parameter named *path metric range (PMR)* to elucidate the properties of different partitions. Then, it is proposed that the list size can change stepwise from one partition to another depending on the average PMR (PMR_{avg}). This stepwise change in the list size (L) throughout decoding is the reason for naming this scheme *stepped list decoding*. The stepped list decoding results in a substantial reduction in memory requirement, as well as the computational complexity of (CRC-aided) SCL decoder. Most notably, these improvements are obtained without sacrificing the error correction performance or introducing an overhead.

Paper Outline: The rest of the paper is organized as follows. Section II introduces the notation for the polar codes and describes the SC and SCL decoding. Section III analyzes the list decoding process and proposes the stepped list decoding scheme. In Section IV, the memory requirement and computation complexity of the stepped list decoder are analyzed. In Section V, the implementation results are shown. Finally Section V makes concluding remarks.

II. Preliminaries

For a polar code $PC(N,K,A)$ where $N = 2^n$ is the block-length, K is the number of information bits and A is the index set of the unfrozen bits, the generator matrix is $G_N = B_N G_2^{\otimes n}$, where $G_2 \triangleq \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, B_N is an $N \times N$ bit-reversal permutation matrix, and $(\cdot)^{\otimes n}$ denotes the n-th Kronecker power [1].

Let $u_0^{N-1} = (u_0, u_1, ..., u_{N-1})$ denote the vector to be encoded, including frozen and unfrozen bits, and $x_0^{N-1} = (x_0, x_1, ..., x_{N-1})$ represent the vector of coded bits given that $x_0^{N-1} = u_0^{N-1} G_N$, while $y_0^{N-1} = (y_0, y_1, ..., y_{N-1})$ denotes the channel output vector in a binary discrete memoryless channel.

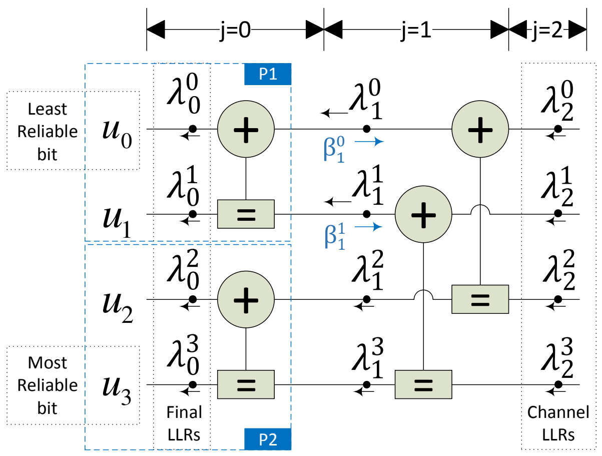


Figure 1. The factor graph of SC decoder for $N = 4$

In SC decoding, the unfrozen bits are estimated successively based on the final LLR via a one-time-pass through the factor graph in Fig. 1 which makes the SC solution sub-optimal. When decoding the i-th bit, if $i \notin A$, $\hat{u}_i = 0$, as u_i is a frozen bit. Otherwise, bit u_i is decided by binary quantizer function $h(\lambda_0^i)$ in (1), which depends on the estimation of previous bits, i.e. $\hat{u}_0, ..., \hat{u}_{i-1}$.

$$\hat{u}_i = h(\lambda_0^i) = \begin{cases} 0 & \lambda_0^i = \ln \frac{P(Y,\hat{u}_0^{i-1}|\hat{u}_i=0)}{P(Y,\hat{u}_0^{i-1}|\hat{u}_i=1)} > 0, \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Unlike SC decoding which follows a certain path at each decision step, successive cancellation list (SCL) decoding follows two paths $u_i = 0$ and $u_i = 1$, when deciding each bit thus SCL considers both possible values 0 and 1. In SCL decoding, the L most reliable paths are preserved at each decoding step. Let $\hat{u}_i[l]$ denote the estimate of u_i in the l-th path, where $l \in \{1, 2, \ldots, L\}$. In [3] unlike [2], a path metric based on LLR magnitudes is used to measure the reliability of the paths. The path metric at $\hat{u}_i[l]$ is approximated by

$$PM_l^{(i)} = \begin{cases} PM_l^{(i-1)} + |\lambda_0^i[l]| & \text{if } \hat{u}_i[l] \neq \frac{1}{2}(1-\text{sgn}(\lambda_0^i[l])) \\ PM_l^{(i-1)} & \text{otherwise} \end{cases} \quad (2)$$

where $PM_l^{(-1)} = 0$.

As (2) shows, the path of the less likely bit value is penalized by λ_0^i of that bit. The L paths with smallest path metrics are chosen from $2L$ paths at the i-th step and stored in ascending order from $PM_1^{(i)}$ to $PM_L^{(i)}$. After decoding the N-th bit, the path with the smallest path metric $PM_1^{(N)}$ is selected as the estimated codeword.

As shown in [2], when the SCL decoder fails, the correct path might still be in the list, when the list size is sufficiently large (e.g., $L = 32$). Adding an r-bit CRC as outer code to the information bits can assist the decoder in finding the correct path among the L paths. However, this concatenation increases the polar code rate to $(K + r)/N$ causing a small performance degradation. This degradation is compensated by the gain obtained at higher SNRs.

III. STEPPED LIST DECODING

A. Analysis of List Decoding

In this section, we investigate the behavior of the list decoder with respect to the evolution of the path metrics within the list throughout decoding a codeword. The relation between this evolution and the likelihood of an error occurrence is empirically analyzed. Then, we will advocate that a fixed list size is not essential, and it can change throughout the decoding process, from one *partition* to another, depending on the properties of such partitions.

Definition 1. *Partitions* [8] are defined as the sub-trees of the decoding tree, associated with code's sub-blocks of length 2^m, $m < n$, that divides the codewords into 2^{n-m} equal-length sub-blocks. The j-th partition and its associated sub-block are denoted by P_j, for $0 \leq j \leq 2^{n-m} - 1$.

In order to characterize the partitions with respect to likelihood of an error occurrence in the list decoding, we introduce a new parameter that helps us to understand the evolution of path metrics throughout the decoding process:

Definition 2. *Path metric range* for the i-th decoding step is defined as $PMR_i = PM_L^{(i)} - PM_1^{(i)}$, where $0 \leq i \leq N - 1$, assuming that the path metrics are sorted in ascending order, i.e. $PM_1^{(i)} < PM_2^{(i)} < \cdots < PM_L^{(i)}$.

Fig. 2 shows the changes of *PMR* throughout the decoding process for $PC(1024, 820)$ (orange curve), along with final LLRs (blue bars) and the frozen bits (red bars). As can be seen, the *PMR* curve elucidates the evolution of the path metrics within the list. Note that the path metric range scales with L; thus, *PMR* reduces if a smaller list size is used.

Now, to analyze the changes in *PMR* value with respect to LLRs, the following lemma is introduced.

Lemma 1. If u_i is an unfrozen bit, i.e. $i \in A$, and $|\lambda_0^i[l]| < PMR_{i-1}$ for all l then $PMR_i < PMR_{i-1}$.

Proof. Assuming $PM_1^{(i-1)} < \ldots < PM_L^{(i-1)}$. After splitting the paths at step/bit i, the $2L$ path metrics are $PM_1^{(i-1)}, PM_1^{(i-1)} + |\lambda_0^i[1]|, \ldots, PM_L^{(i-1)}, PM_L^{(i-1)} + |\lambda_0^i[L]|$. The relation $PM_l^{(i-1)} < PM_l^{(i-1)} + |\lambda_0^i[l]|$ holds for $l = 1, 2, ..., L$. Thus, considering $|\lambda_0^i[l]| < PMR_{i-1}$, then $PM_1^{(i-1)} + |\lambda_0^i[1]| < PM_L^{(i-1)}$. Therefore all the paths greater or equal to $PM_L^{(i-1)}$ are pruned in order to make room for at least the new path $PM_1^{(i-1)} + |\lambda_0^i[1]|$. As a result, $PM_L^{(i)} < PM_L^{(i-1)}$, then $PMR_i < PMR_{i-1}$. ■

As a result of Lemma 1, a subsequence of k bits such that $|\lambda_0^m[l]| < PMR_{m-1}$, for all l and $m = i \ldots i + k$ leads to a sharp drop of the PM range i.e. $PMR_{i+k} \ll PMR_i$.

Here, let us distinguish the bit-channels causing the *PMR* drop by the following definition:

Definition 3. *Crucial bits* are defined as the unfrozen bits with $|\lambda_0^i[l]| < PMR_{i-1}$. S^j denotes the set of indices of crucial bits in the j-th partition.

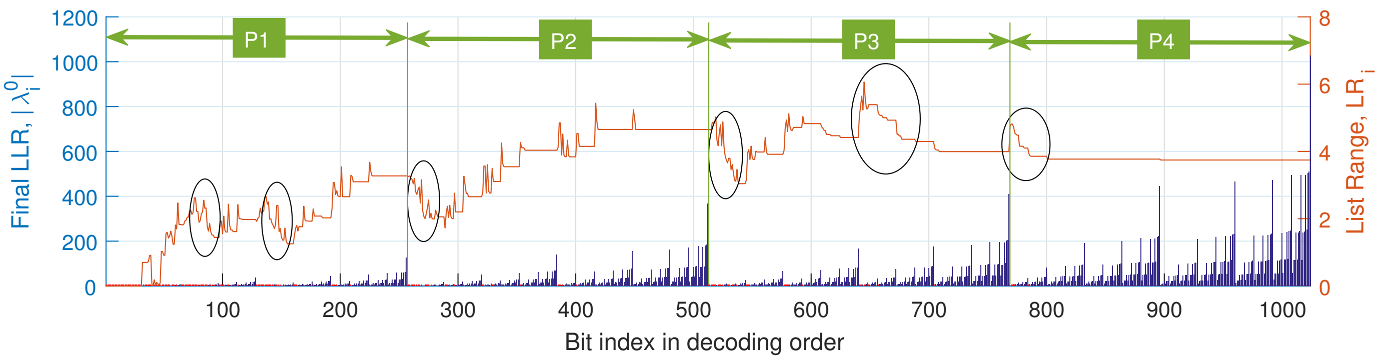


Figure 2. Absolute values (average in 100 iterations) of bit-channel LLRs $|\lambda_0^i|$ and path metric range (LR_i), in natural decoding order for PC(1024,820)

The local minima on the *PMR* curve (orange) in Fig. 2 appear after a series of crucial bits (see circled sections in Fig. 2) where most of the errors are observed.

From Lemma 1 and its following discussion, one can infer that if u_i and u_j are crucial bits (possibly in different partitions) and $PMR_i \ll PMR_j$, there exists a list size L', $L' < L$, so that $|\lambda_0^j[l]| < PMR'_j$ still holds for all l. Note that since *PMR* scales with L, then $PMR'_j < PMR_j$. Here, L is the initial list size and PMR' is the new path metric range obtained after reducing the list size to L'. This leads us to the conclusion that reducing the list size in the partition with higher minimum *PMR* does not affect the error correction performance, if the new list size for that partition is chosen optimally.

As a practical method, the inverse of the average PMR over the crucial bits in j-th partition $PMR_{avg}^{(j)}$ can be employed to determine the local list size L_j, i.e.

$$\log_2(L_j) \propto \frac{1}{PMR_{avg}^{(j)}} = \frac{|S^j|}{\sum_{S^j} PMR_i} \ .$$

For instance, the 4-tuple of average *PMR* (rounded) over crucial bits in the four partitions of the example shown in Fig. 2 is $(PMR_{avg}^{(j)})_{1\leq j\leq 4} = (2,3,4,4)$. Thus, the list size for the four partitions are obtained by the mapping $(2,3,4,4) \rightarrow (2^t, 2^{t-1}, 2^{t-2}, 2^{t-2})$, where the maximum list size ($L = 2^t$) is assigned to the partition with the smallest PMR_{avg}. Alternatively, if we use the *PMR* curve as a graphical tool, the maximum list size is assigned to the partition in which the global minimum of *PMR* curve is located. The list size for the rest of the partitions are assigned with respect to the local minima of *PMR*. Note that the *PMR* curve changes for different code lengths and code rates.

Now, we describe the impact of *PMR* on the possibility of error occurrence. Assuming l_c denotes the index of the correct path. In the list decoding process, the correct path may not always correspond to the smallest path metric $PM_1^{(i)}$. Due to the penalties, it may have a larger path metric $PM_{l_c}^{(i)} > PM_1^{(i)}$.

Fig. 3 shows different scenarios for the movement of the correct path (with index l_c) within the list throughout list decoding. The frequent penalty scenario mainly occurs in the partition(s) in which the local PMR_{avg} is smaller. The olive and orange curves in Fig. 3 show the changes in l_c in the indices below 300 which are located in P_1 and P_2. In the scenario shown by olive curve, the correct path is pruned after several penalties, while the orange curve shows the scenario where the correct path remains in the list.These scenarios show that the list size in P_1 should be large enough to retain the correct path within the list in case of bearing frequent penalties while in the subsequent partitions P_2 and P_3 where PMR_{avg}s corresponding to crucial bits gradually increases, the list size can be reduced without any significant effect on the error correction performance.

In the last partition, since $PMR_{avg}^{(4)}$ is relatively large, if the correct path is penalized over some crucial bits, the path will remain in the list due to low number of crucial bits in that partition and consequently low probability of frequent penalties. Note that the index of penalized path does not increase significantly when L is large. The blue curve in Fig. 3 illustrates this scenario.

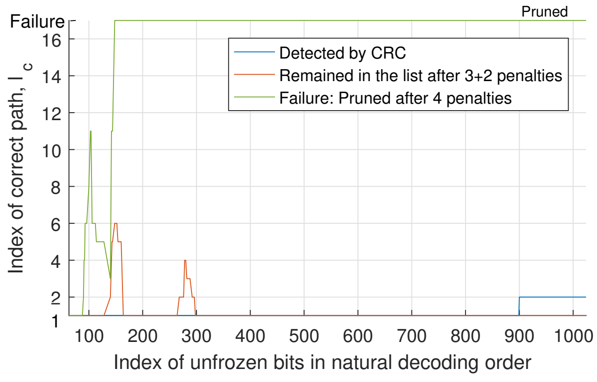


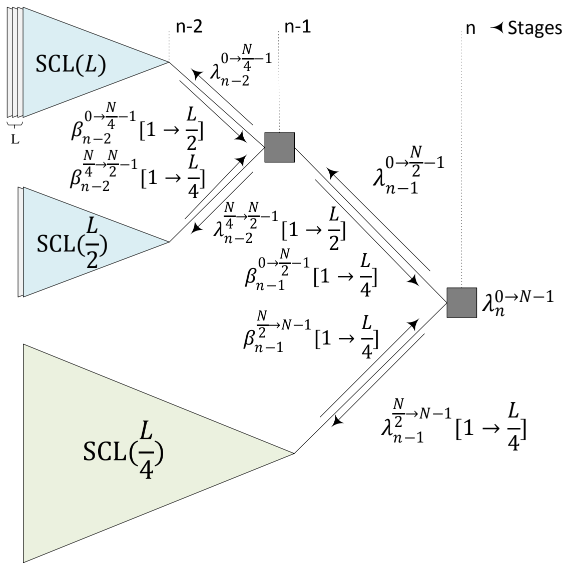
Figure 3. Some sampled movements of correct path in the list, representing different scenarios, for PC(1024,820) and $L = 16$

B. The Stepped List Decoding

Since the average path metric range (PMR_{avg}) varies from one partition to another, the fixed list size (L) in conventional list decoding is mainly effective on the partition(s) with the relatively small PMR_{avg}. In the partition(s) with significantly larger PMR_{avg}, the potential of the L is not fully used. Hence, we can allocate different list sizes to different partitions based on PMR_{avg}. In this scheme, the list size changes stepwise from one partition to another and that is the reason for calling it *stepped list decoding*. The effective list size for partitions are allocated based on PMR_{avg}. The computed PMR_{avg} for all the partitions are clustered with respect to a significant difference among them. For instance, the 4-tuple of rounded PMR_{avg} for four partitions of PC(1024,256) at 1dB is (-,38,23,24), where 23 and 24 will be in one cluster and therefore an identical list size will be assigned to both P_3 and P_4. Thus, the list size mapping could be $(-, 38, 23, 24) \to (2, 2^{t-1}, 2^t, 2^t)$. As another example, since the 4-tuple of rounded PMR_{avg} for four partitions of PC(1024,512) at 1.4dB is (8,6,6,9), the list sizes could be allocated as $(2^{t-1}, 2^t, 2^t, 2^{t-2})$. It will be seen in section V that by proper allocation of list sizes to the partitions, the error correction performance will not degrade.

Note that in this paper, although the examples are based on four partitions, the number of partitions could be larger.

Although the memory requirement differs from one code rate to another in stepped list decoding, the largest memory requirement can be used in multi-mode scheme, where different settings are employed by changing the code parameters.



n-2
n-1
n
Stages
SCL(L)
L
SCL($\frac{L}{2}$)
SCL($\frac{L}{4}$)
$\lambda_{n-2}^{0\to\frac{N}{4}-1}$
$\beta_{n-2}^{0\to\frac{N}{4}-1}[1\to\frac{L}{2}]$
$\beta_{n-2}^{\frac{N}{4}\to\frac{N}{2}-1}[1\to\frac{L}{4}]$
$\lambda_{n-2}^{\frac{N}{4}\to\frac{N}{2}-1}[1\to\frac{L}{2}]$
$\lambda_{n-1}^{0\to\frac{N}{2}-1}$
$\beta_{n-1}^{0\to\frac{N}{2}-1}[1\to\frac{L}{4}]$
$\beta_{n-1}^{\frac{N}{2}\to N-1}[1\to\frac{L}{4}]$
$\lambda_{n}^{0\to N-1}$
$\lambda_{n-1}^{\frac{N}{2}\to N-1}[1\to\frac{L}{4}]$

Figure 4. Stepped list decoding tree for PC(1024,820)

IV. Analysis of Computational Complexity and Memory Requirement

The stepped list decoding proposed in section III requires significantly less memory space and less computations than conventional (CRC-aided) SC List decoding process. In this section, we investigate the impact of the stepped list decoding on computational complexity and memory requirement for two examples discussed in the previous section, PC(1024,820) and PC(1024,820), where the 4-tuples of the list sizes for four partitions are $(L, L/2, L/4, L/4)$ and $(L/2, L, L, L/4)$, respectively, instead of fixed list size L for all the partitions.

A. Computational Complexity

Consider the computational complexity of the conventional list decoding, $L \cdot N \log_2 N$. Since the list size in the stepped

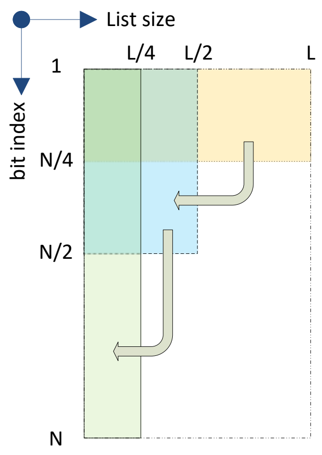


Figure 5. Sketch of path memory in stepped list decoding of PC(1024,820)

list decoding changes, the computational complexity changes to (3), showing that the complexity reduces 50% in 3-step scheme (Fig. 4).

$$\underbrace{L\frac{N}{4}\log_2 N}_{\text{1st partition}} + \underbrace{\frac{L}{2}\frac{N}{4}\log_2 N}_{\text{2nd partition}} + \underbrace{\frac{L}{4}\frac{N}{2}\log_2 N}_{\text{3rd/4th partition}} = \frac{1}{2}LN\log_2 N \quad (3)$$

Similarly, the computational complexity for PC(1024,512) with the 4-tuple of list sizes $(L/2, L, L, L/4)$ can be derived as $\frac{21}{32}LN\log_2 N$, which is 34% less than conventional list decoding process.

Note that the reduction in complexity is SNR-independent unlike the tree/list pruning techniques.

B. Memory Requirement for Candidate Paths

Similar to computational complexity, the memory required for storing the estimated bits of the candidate paths is directly proportional to L. The partitioning helps to allocate the path memory efficiently. As the decoding proceeds from the last bit of one partition to the first bit of the next partition, since L is halved, half of the allocated memory is freed, as shown in Fig. 5. This freed space can be used for the next partition.

As a result, the memory reduction for the proposed 3-step scheme for PC(1024,820) is $L \cdot N - L/4 \cdot N$ bits or 75%.

Similarly, the path memory requirement for PC(1024,512) with the 4-tuple of list sizes $(L/2, L, L, L/4)$ can be derived as $\frac{3}{4}LN$, which is 33% less than conventional list decoding.

C. Memory Requirement for LLRs and Partial Sums

The memories required for internal LLRs and partial sums are proportional to the list size, similar to the path memory. In the conventional list decoding, the internal LLRs, $\lambda_{0\to n-1}^{0\to N-1}$, need $(N-1)\cdot L\cdot Q$ bits and the partial sums, $\beta_{0\to n-1}^{0\to N-1}$, require $(N-1)\cdot L$ bits [9]. We know that $N/2 \cdot L$ and $N/4 \cdot L$ out of $(N-1)\cdot L$ memory elements are allocated to stage $n-1$ and stage $n-2$, respectively. However, in the stepped list decoding for PC(1024,820), since the list size in every subsequent partition is halved, only half of the partial sums of preceding stage are sent backward. The aforementioned process is depicted in Fig. 4 by an index range in the bracket. Note that no bracketed index ranges have been added to λ_{n-1} of bit-channels 0 to $\frac{N}{2}-1$ and λ_{n-2} of bit-channels 0 to $\frac{N}{4}-1$ because they are not dependent on partial sums; therefore, they are the same for all paths in the list.

According to the above discussion for PC(1024,820), the memory requirement for LLRs and partial sums can reduce from (4) in the conventional list decoding, to (5) in the stepped list decoding.

$$M_{SCL} = \underbrace{L(N-1)Q_i}_{\text{Internal LLRs}} + \underbrace{L(N-1)}_{\text{Partial sums}} \tag{4}$$

$$\begin{aligned} M_{SCL-Stepped} = & \underbrace{\Big(L(\frac{N}{4}-1) + \frac{L}{2}(\frac{N}{4}) + \frac{L}{4}(\frac{N}{2})\Big)Q_i}_{\text{Internal LLRs}} \\ & + \underbrace{\Big(L(\frac{N}{4}-1) + \frac{L}{2}(\frac{N}{4}) + \frac{L}{4}(\frac{N}{2})\Big)}_{\text{Partial sums}} \\ & = \underbrace{L(\frac{1}{2}N-1)Q_i}_{\text{Internal LLRs}} + \underbrace{L(\frac{1}{2}N-1)}_{\text{Partial sums}} \end{aligned} \tag{5}$$

where Q_i is the number of bits used in quantization of internal LLRs. Note that in all cases, channel LLRs require an additional memory of NQ_{ch} bits, where Q_{ch} is the number of quantization bits for channel LLRs. This is omitted from the above equations for simplicity.

By comparing (5) with (4), it is concluded that the memory reduction using stepped list decoding is 50% in the proposed 3-step scheme for PC(1024,820).

Similarly, the memory required for internal LLRs and partial sums of PC(1024,512) with the 4-tuple of list sizes $(L/2, L, L, L/4)$ can be derived as $L(\frac{21}{32}N-1)Q_i + L(\frac{21}{32}N-1)$, which is 34% less than the conventional list decoding.

V. SIMULATION RESULTS

The LLR-based stepped CA-SCL decoder is implemented for polar codes of $N = 2^{10}$ and the code rates $R = K/N = 0.8$ and 0.5 over AWGN channel. The polar codes are constructed using Bhattacharya parameter (heuristic) method and optimized for high SNRs. The 16-bit CRC generator polynomial $g(x) = x^{16}+x^{12}+x^5+1$ is used for correct path detection. For step list decoding of PC(1024,820) and PC(1024,512), the list size 4-tuples $(32, 16, 8, 8)$ and $(16, 32, 32, 8)$, respectively, are used for the partitions. Fig. 6 compares the performance of conventional CRC-aided SCL decoder with the proposed one. The proposed stepped list decoding preserves the performance of conventional list decoding with fixed list size in various code rates.

VI. CONCLUSION

In this paper, we analyze the list decoding process by introducing a new parameter named path metric range and tracking the correct path within the list with respect to value of this parameter. Then, we propose a complexity-reduced memory-efficient list decoder in which the list size changes in the subsequent partitions of the decoding tree. The results of simulations for polar codes of length 1Kb and $R = 0.5$ and

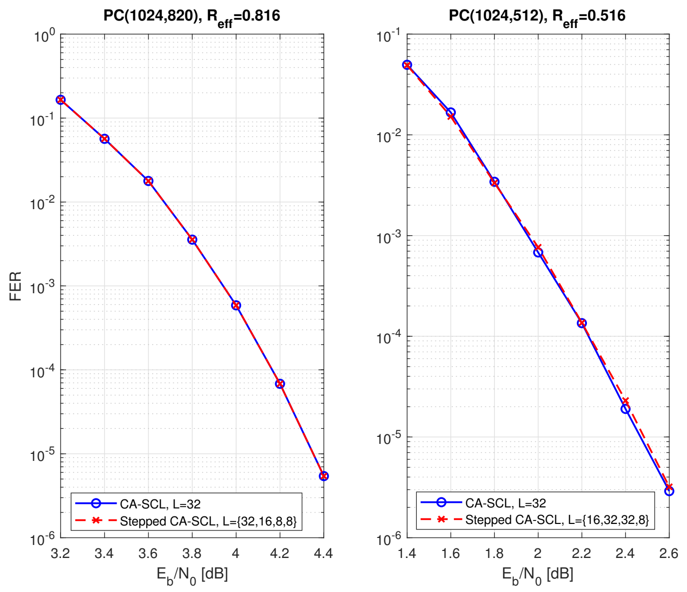


Figure 6. Performance of the Stepped CA-SCL vs CA-SCL Decoding

0.8 show that the performance of the conventional list decoder in the stepped list decoding scheme is preserved. However, the stepped SCL decoding maximally can reduce the path memory by 75% and LLRs memory and partial sums memory as well as computational complexity by 50%. The stepped list decoding can be used in the partitioned SCL decoding to reduce the complexity, and in tree-pruning schemes to lower the memory requirement.

ACKNOWLEDGMENT

The authors would like to thank Dr. Lilian Khaw and Fariba Abbasi for their comments on the paper.

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," *IEEE Int. Symp. on Information Theory,* St. Petersburg, Russia, Jul. 2011, pp. 1–5.
[3] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Trans. Signal Processing,* vol. 63, no. 19, pp. 5165-5179, Oct 2015.
[4] K. Chen, B. Li, H. Shen, J. Jin, and D. Tse, "Reduce the complexity of list decoding of polar codes by tree-pruning," *IEEE Commun. Lett.,* vol. 20, no. 2, pp. 204-207, Feb. 2016.
[5] J. Chen, Y.Z. Fan, C.Y. Xia, C.Y. Tsui, J. Jin, K. Chen, and B. Li, "Low-Complexity List Successive-Cancellation Decoding of Polar Codes Using List Pruning," *IEEE Global Communications Conference,* Washington DC, USA, Dec. 2016, pp. 1-6.
[6] Z. Zhang, L. Zhang, X. Wang, C. Zhong, H. V. Poor, "A split-reduced successive cancellation list decoder for polar codes," *IEEE J. Select. Areas Commun.,* vol. 34, no. 2, pp. 292-302, Feb. 2015.
[7] J. Guo, Z. Shi, Z. Liu, Z. Zhang, Q. Liu, "Multi-CRC polar codes and their applications," *Commun. Lett.,* vol. 20, no. 2, pp. 212-215, Feb. 2016.
[8] S. A. Hashemi, A. Balatsoukas-Stimming†, P. Giard, C. Thibeault, and W. J. Gross, "Partitioned Successive-Cancellation List Decoding of Polar Codes," *IEEE Inter. Conf. on Acoustics Speech and Sig. Process. (ICASSP),* Shanghai, China, March 2016, pp. 957-960.
[9] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Trans. Signal Process.,* vol. 61, no. 2, pp. 289-299, Jan. 2013.