Complexity-adjustable SC decoding of polar codes for energy consumption reduction

Citation for published version (APA):

DOI:
10.1049/iet-com.2018.5643

Document status and date:
Published: 27/08/2019

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Complexity-adjustable SC decoding of polar codes for energy consumption reduction

Haotian Zheng1, Bin Chen1,2, Luis F. Abanto-Leon1,3, Zizheng Cao1 5, Ton Koonen1

1Eindhoven University of Technology, NL 5600 MB Eindhoven, The Netherlands
2School of Computer and Information, Hefei University of Technology (HFUT), Hefei, People’s Republic of China
3Department of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany
E-mail: z.cao@tue.nl

Abstract: This study proposes an enhanced list-aided successive cancellation stack (ELSCS) decoding algorithm with adjustable decoding complexity. Also, a logarithmic likelihood ratio-threshold based path extension scheme is designed to further reduce the computational complexity. A stack decoding. Numerical simulation results show that without affecting the error correction performance, the proposed ELSCS decoding algorithm provides a flexible trade-off between time complexity and computational complexity, while reducing storage space up to 70%. Based on the fact that most mobile devices operate in environments with stringent energy budget to support diverse applications, the proposed scheme is a promising candidate for meeting requirements of different applications while maintaining a low computational complexity and computing resource utilisation.

1 Introduction

With the ever-increasing number of mobile communication devices and amount of communication data, power consumption becomes a tremendous challenge for mobile communication devices, especially for those operating under very stringent energy budget. To obtain high energy efficiency, the high energy-consuming error-correcting code decoder is a pivotal component in the communications chain that needs to be suitably designed. However, different applications have a diverse set of communication requirements, an one-size-fits-all energy-efficiency decoder is unfeasible. Therefore, a flexible decoder that can adjust the trade-off between the conflicting demands for energy, throughput and error rate performance is essential for achieving low energy expenditure in different applications.

Considering its explicit decoding construction and low decoding complexity, polar codes are strong candidates to fulfill such requirements. These codes were proposed by Arikan and are the first provable capacity-achieving code for any binary-input discrete memoryless channels [1, 2]. Polar codes have been shown to outperform LDPC code at moderate and short code length [3, 4]. In the newest 5G coding standard, polar codes have been adopted as channel coding for control channels in the enhanced mobile broadband communications services [5].

The most widely used high-performance decoding algorithm for polar codes is CRC-aided successive cancellation list (CA-SCL) decoding [3, 6]. Nevertheless, it still requires a large list size to achieve a good error rate performance, which inevitably leads to even higher computational complexity. Another drawback is that the complexity of CA-SCL remains constant in different signal-to-noise ratio (SNR) regimes when in fact, fewer computations are required for correct decoding in the high SNR regime, thus causing unnecessary energy expenditure.

Previous works have studied the simplified calculation of successive cancellation (SC)/successive cancellation list (SCL) decoding [7–11]. They found removable and replaceable redundant calculations when encountering frozen bit and particular information patterns, while keeping the error rate performance unaltered. With a similar principle, a new class of relaxed polar codes was introduced by deactivating polarisation units when the bit-channels are sufficiently good or bad [12]. Irregular polar codes generalised this idea to further consider all possible selections of the inactivated polarisation units and obtained a significant decoding complexity reduction [4].

In addition to the simplification of the decoding process, path pruning is also an efficient way to reduce the computational complexity of SCL decoding. Chen et al. [13, 14] proposed to set up a flexible path metric threshold to prune decoding paths with small path metric. In [15], the concept of relative path metric (RPM) was introduced and the correct candidate was found to have a lower probability of having a large RPM value. Paths which do not satisfy this property are pruned and the computational complexity can thereby be effectively decreased.

Adaptive SCL algorithm was introduced in [16]. It sets the initial value of the list size to 1. If the decoding attempt succeeds, the decoding process stops and the decoding result is output. Otherwise, the list size is doubled and the SCL decoding algorithm is restarted. This procedure is repeated until the list size reaches a predefined maximum threshold. If the decoding attempt still fails, a decoding failure is declared. This tentative early-stopped strategy enables adaptive SCL algorithm to have SNR-adaptive capability. Furthermore, this kind of strategy is also applicable to other decoding algorithms. As long as the algorithm itself has lower computational complexity than SCL, the resultant decoding complexity of the new hybrid using this strategy will also be lower.

In fact, when decoding latency is not so stringent and throughput demand is not very high, another successful cancellation stack (SCS) decoding algorithm [17] is a more energy-efficient choice than SCL. Using trading storage complexity for computational complexity reduction, SCS has a much lower and SNR-adaptive computational complexity. It was employed inside the kernel processor to reduce substantially the decoding complexity of polar codes with Reed–Solomon kernel [18]. In addition, the efficient software implementation of the SCS decoding algorithm has been investigated [19]. Moreover, the computational complexity reduction schemes mentioned above are equally applicable to SCS, and can further decrease the required amount of computations. However, lower energy consumption leads to the disadvantage of smaller throughput. Because of employing breadth-first instead of depth-first search, the decoding delay of SCS will be larger than that of SCL with the same error rate performance, especially in the low SNR regime.

From the discussions above, we can observe that SCL and SCS have complementary performance. This inspired us to combine the advantages of both and propose an adjustable decoder which can make a flexible trade-off between energy consumption and throughput for a certain given error rate performance as shown in Fig. 1. The main contributions of this work are described follows:

© The Institution of Engineering and Technology 2019
2 Preliminaries

2.1 Notation conventions

In this paper, we use the notation $a^i_j$ to represent the vector $(a_i, a_{i+1}, \ldots, a_j)$ and write $a^e_i, a^o_i$ to denote the subvector of $a_i^j$ with even and odd indices, respectively. For a positive integer $m$, $[m] \triangleq \{1, 2, \ldots, m\}$.

Only square matrices are used in this paper and $F_N$ represents an $N \times N$ matrix $F, (F_N)^\otimes n$ denotes the $n$th Kronecker power of $F$.

Letters $\mathcal{A}$ and $\mathcal{B}$ stand for memory stacks and $[\mathcal{A}]$ denotes the number of elements in $\mathcal{A}$. Letter $\gamma$ denotes the energy per bit-to-noise power spectral density ratio ($E_b/N_0$). Throughout this paper, $\ln(\cdot)$ represents the natural logarithm and $\log(\cdot)$ indicates ‘logarithm to base 2’.

2.2 Polar codes

Let $W: x \rightarrow y$ denote a binary discrete memoryless channel with input alphabet $\mathcal{X} = \{0, 1\}$, output alphabet $\mathcal{Y}$ and channel transition probabilities $\{W(y|x) : x \in \mathcal{X}, y \in \mathcal{Y}\}$. The channel polarisation operation in Fig. 2 takes place recursively on $N$ independent copies of channel $W$. After channel polarisation, the transition probability of the $i$th synthesised subchannel is given by

$$W^0_N(y^N_i, u^i_{-1} | u_i) = \sum_{u^i_1, \in \mathcal{X}^{-i}} \frac{1}{2^{N-1}} W_N(y^N_i | u^N_i)$$

where

$$W_N(y^N_i | u^N_i) = \prod_{i=1}^{N} W(y|x_i)$$

Although the total channel capacity remains unchanged after channel polarisation, the capacity of synthesised subchannels has been polarised. It was proven by Arikan [1] that as block-length approaches infinity, a portion of subchannels attains a capacity close to 1, whereas the capacity of the remaining subchannels isproximate to 0. The subchannels with higher quality are used for sending information bits, while the remaining ones are assigned fixed bits that are known to both sender and receiver in advance. This improves the reliability of useful information transmission and becomes the theoretical basis of polar codes.

For a polar code with block-length $N = 2^n$ and rate $R$, the generator matrix can be written as $G_n = B_n G^\otimes n$, where

$$G_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$
where $\Xi$ is defined as $\ln(1 + e^{\alpha + \beta})/(e^\alpha + e^\beta)$ and can be approximated as a more hardware-friendly function $\operatorname{sign}(\alpha)\operatorname{sign}(\beta)\min\{|\alpha|, |\beta|\}$.

SC decoding employs sequential decoding. If $\hat{u}_i$ is a frozen bit, its estimated value is directly set to zero. Otherwise, the estimated value is given by

$$\hat{u}_i = \begin{cases} 0 & \text{if } L_{i[N]}^\alpha \geq 0 \\ 1 & \text{if } L_{i[N]}^\alpha < 0 \end{cases} \quad (5)$$

Unlike the SC decoder, which uses hard-decision for each bit and has only one search path, SCL and SCS extend the decoding path to two new paths by appending a bit 0 or a bit 1 when an unencoded bit is encountered. Figs. 3 and 4 show simple examples of the tree search process of SCS and SCL decoding algorithms, respectively. The bold branches represent the decoding path. The number next to each node is the maximum metric of the decoding path from the root to that node. The nodes generated after path extensions are represented by the numbered circles with the numbers indicating the extension stage order and the node with red number is the correct decoding path. The grey ones are nodes that are not visited during the search process.

SCL will firstly sort the new generated paths by their path metrics, which is defined by

$$\text{PM}_i = \ln(P(\hat{u}_i | Y)) \quad (6)$$

PM$_i$ represents the path metric when the decoding length is $i$, whereas metric will be used as an abbreviation for path metric in the following. Thereupon, a maximum of $L$ ($L$ denotes the list size) paths with the same length and the largest metrics are selected for the next path extension stage without storing other paths. The decoding length is increased by one at each path extension stage. When it reaches the block-length $N$, the path with the maximum metric is output as the decoding sequence. SCS stores all generated paths in a stack and selects the top path with the maximum metric to extend each time. An additional parameter, search width $Q$, is added to limit the number of extending paths with certain lengths in the decoding process. Whenever the top path with the largest metric in the stack reaches block-length $N$, the decoding process stops and outputs this path. Compared with SC, SCL and SCS have much wider search range and less probability to fall into a local optimum, which certainly ensures an improved error rate performance.

As SCL is a depth-first search algorithm, it extends $L$ paths with the largest metrics at the same length each time and the decoding length keeps increasing. Different from SCL, SCS employs breadth-first search, thus selecting only one path with the maximum metric to extend each time. The decoding path length in SCS is not always increasing after each extension. Therefore, it takes the decoding path a long time to reach the block-length $N$. This explains why SCS has a larger decoding latency compared to SCL. As shown in the examples, SCL only needs to extend four times to complete the decoding while SCS needs two more. Another difference is that SCS uses a stack to store some candidate partial decoding paths with different lengths. In case of good channel quality, the correct path has a relatively larger metric than incorrect ones. For the sake of using a stack, SCS can selectively choose a candidate partial path with a large metric to search instead of searching $L$ paths blindly like SCL. Therefore, compared to SCL, SCS with the same error rate performance (when $Q = L$), a lot of unnecessary searches are avoided. The computational complexity of SCL is approximately $L \cdot N \cdot \log N$ and that of SCS is variable and much less than that of the former. We can see from the two examples, five path extension operations are saved by SCS compared to SCL.

In CRC-aided decoding, the source information contains a built-in CRC and it chooses the path with the maximum metric from those who pass CRC check as output when the decoding length reaches $N$. In the case of poor channel quality, the correct path does not necessarily have the largest metric, but certainly passes the CRC check. For this reason, CRC aided decoding can further improve the probability of successful decoding.

### 3 LLR-threshold based path extension scheme

To combine the advantages of SCS and SCL, we need to adopt both stack decoding and list decoding. However, in stack decoding, much space is occupied by the stack to store candidate partial decoding paths with different lengths. If list decoding is further considered, more candidate partial paths need to be stored, which will occupy considerable device memory. Aiming at decreasing the required space, we propose an LLR-threshold based path extension (LLPTE) scheme. Before illustrating it, a conjecture is given, which will provide the theoretical basis for the scheme. If the decoding bit of the correct path is an unfrozen bit, the below phenomenon happens with a large probability. Among the two generated paths after path extension, the metric of one path remains almost unchanged and that of the other path will decrease a lot. The higher the reliability of the corresponding polarisation channel, the greater the probability of this phenomenon. Next, we will give empirical evidence of this conjecture.

The update function of the metric used for path selection is

$$\text{PM}_i(\epsilon) = \text{PM}_{i-1}(\epsilon) - \ln(1 + e^{-\lambda - 2\mu \epsilon} \cdot e^{\lambda L_{i[N]}^\epsilon}) \quad (7)$$

[6]. $\text{PM}_i(\epsilon)$ represents the metric of path $\epsilon$ with length $i$. $\hat{u}_i(\epsilon)$ represents the estimated value of the $i$th bit of path $\epsilon$. $L_{i[N]}^\epsilon$ represents $L_{i[N]}^\epsilon$ of path $\epsilon$. The update function can be expressed as

$$\text{PM}_i(\epsilon) = \phi(\text{PM}_{i-1}(\epsilon), L_{i[N]}^\epsilon, \hat{u}_i(\epsilon)),$$

where the function $\phi(\mu, \lambda, u) \triangleq \mu - \ln(1 + e^{-(\lambda - 2\mu)u})$ can be approximated to

$$\phi(\mu, \lambda, u) \triangleq \begin{cases} \mu & \text{if } u = \frac{1}{2}(1 - \operatorname{sign}(\lambda)) \\ \mu - |\lambda| & \text{otherwise} \end{cases} \quad (8)$$

**Fig. 3** Example of the SCL decoding with $L = 4$

**Fig. 4** Example of the SCS decoding
Thus, the update function can be approximately recast as
\[
\hat{\text{PM}}_t(\ell) \triangleq \begin{cases} 
\text{PM}_t(\ell) & \text{if } \hat{u}_t(\ell) = \frac{1}{2} \left[ 1 - \text{sign} \left( L_0^t(\ell) \right) \right] \\
\text{PM}_t(\ell) - [L_0^t(\ell)] & \text{otherwise}
\end{cases} 
\] (9)

Next, we use \( L_t \) to represent \( L_0^t \) of the correct path and analyse the value of \( L_t \). Consider a Gaussian channel, under the assumption of Gaussian approximation, \( L_t \sim \mathcal{N} \left( \text{E}[L_t], 2 \text{E}[L_t^2] \right) \) [24], where
\[
\text{E}[L_t] = \begin{cases} 
\text{E}[L_0(0)] & \text{if } u_t = 0 \\
-\text{E}[L_0(0)] & \text{if } u_t = 1
\end{cases} 
\] (10)

\( \text{E}[L_0(0)] \) denotes the expectation of \( L_0 \) for the all-zero codeword transmitted over a binary-input additive white Gaussian noise channel (BI-AWGN) with binary phase shift keying (BPSK) modulation and noise variance \( \sigma^2 \), which can be calculated recursively by the following equations:
\[
\begin{align*}
\phi(x) &= \begin{cases} 
1 - \frac{1}{\sqrt{4\pi}} \int_x^\infty \frac{1}{\sqrt{u}} \exp \left( -\frac{(u-x)^2}{4u} \right) \, du & x \neq 0 \\
0 & x = 0
\end{cases} 
\end{align*} 
\] (12)

Given \( \delta\text{er}_{\gamma}, \text{E}[L_0(0)] \) and \( \text{E}[L_t] \) can be calculated in an off-line manner.

The relationship between the reliability of polarised channel and \( \text{E}[L_t] \) is shown as below:
\[
P_e(u_t) = \int_0^\infty \frac{1}{2\sqrt{\pi\text{E}[L_t]}} \cdot \exp \left( -\frac{(x - \text{E}[L_t])^2}{4\text{E}[L_t]} \right) \, dx = Q(\sqrt{\text{E}[L_t]/2}) 
\] (13)

where \( P_e(u_t) \) represents the probability that \( u_t \) is incorrectly estimated in terms of the subchannel \( W_0^t(\gamma, u_t^{-1} \lvert \alpha) \), given the correct prior bits \( u_t^{-1} \). The smaller \( P_e(u_t) \), the higher the channel reliability. \( Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} \, dt \) is a monotonically decreasing function, thus the channel reliability increases with the increasing value of \( \text{E}[L_t] \).

The unfrozen bits are chosen according to the channel reliability and corresponds to the \( N \times R \) polarisation channels with the smallest \( P_e(u_t) \) and the largest \( \text{E}[L_t] \). Consequently, for an unfrozen bit \( i \), \( L_t \) has a high probability to be a large value, and the greater reliability polarisation channel \( i \) has, with higher probability this happens. We can see from Fig. 5 that when \( N = 1024, \gamma = 2 \) dB, the smallest \( \text{E}[L_t] \) for unfrozen bits is 14.2, the values at some other \( \gamma \) are also shown in Table 1. When the correct decoding path comes to unfrozen bit \( i \) in sequential decoding, the value of \( L_t \) is large, especially when the corresponding polarisation channel has high reliability. Consider (9), it happens with the large probability that the metric of one path remains almost unchanged after path extension and another will decrease considerably.

The larger the metric gap, the smaller the probability that the latter one to be selected as the final decoding path. These kinds of paths occupy a lot of storage space. Hence, we consider setting an LRT threshold. If the metric difference is larger than the threshold, the one with smaller metric will be deleted. The other one whose metric almost remains constant will continue to be extended at the next path extension stage and does not need to be considered in the sorting stage. Let \( \delta = \delta^* \) denote the LRT threshold. The LTP extension scheme is summarised in Algorithm 1 (see Fig. 6).

In Algorithm 1 (Fig. 6), \( L_{t+1}(\ell) \) represents \( L_t^t \) of the decoding path \( \ell \), \( (u_t(\ell), u_t^{-1}(\ell)) \) denotes the generated new path after path \( \ell \) chooses \( u_t^{-1} \) to extend at the \( t \)th bit. This LTP strategy can reduce the number of paths inserted into the stack, avoiding the excessive space occupied by those paths that have a small probability of being the final decoding output. As fewer bits are required for the quantisation of LRT than metric in hardware implementation, it is easier to compare LRT than metric [6]. Also, the deleted paths in the LRT-based scheme are not required to execute (9) for metric computation. Hence, the proposed LRT-based path pruning scheme is more convenient to implement in practice, compared with existing metric-based path pruning schemes. Of course, we can also use metric-based schemes after the proposed LRT-based scheme to further lower the memory requirements, at the cost of higher implementation complexity.

For how to suitably select the threshold \( \delta \) to reduce the storage requirements while maintaining error rate performance unaffected, we will give an in-depth analysis and discussion in the simulation part in Section 6.

![Fig. 5](image-url)  
**Fig. 5** \( \text{E}[L_t] \) with different indices

<table>
<thead>
<tr>
<th>( \gamma ) (( N = 1024 ))</th>
<th>1.0 dB</th>
<th>1.5 dB</th>
<th>2.0 dB</th>
<th>2.5 dB</th>
<th>3.0 dB</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \min(\text{E}[L_t]) ), ( i ) is unfrozen bit</td>
<td>5.38</td>
<td>9.38</td>
<td>14.20</td>
<td>23.00</td>
<td>30.00</td>
</tr>
</tbody>
</table>

1. \( A \) denotes the stack for storing paths to be extended;  
2. \( B \) denotes the stack for storing generated paths after extension;  
3. if \( u_{t+1} \) is unfrozen bit then  
4. if \( |L_{t+1}(0)| \geq \delta \) then  
5. \( \ell \) Delete \( (u_t^t(\ell), u_{t+1}^{-1}) \), \( u_{t+1}^{-1} \neq \frac{1}{2} \left[ 1 - \text{sign}(L_{t+1}^{-1}(0)) \right] \);  
6. Reserve \( (u_t^t(\ell), u_{t+1}^{-1}) \), \( u_{t+1}^{-1} = \frac{1}{2} \left[ 1 - \text{sign}(L_{t+1}^{-1}(0)) \right] \) in stack \( A \) for next path extension stage directly;  
7. else  
8. Push both paths into stack \( B \) for sorting process;  
9. end  
10. else  
11. Push \( (u_t^t(\ell), 0) \) into stack \( B \) for sorting process;  
12. end

![Fig. 6](image-url)  
**Fig. 6** Algorithm 1: LRT-threshold based path extension scheme

---

© The Institution of Engineering and Technology 2019

---

4

---

© The Institution of Engineering and Technology 2019
Define $Q$ as the maximum tolerable number of CRC check;
Define $L$ (1 ≤ $L$ ≤ $Q$) as the number of paths that are extended simultaneously at each extension stage;

1) Initialization
4. Create a stack $A$ with depth $L$;
5. Create a stack $B$ with depth $D$;
6. Push a null path into $A$, set path metric to 0;
7. Initialize counter, $q_j^N = \text{zeros}(N, 1)$;

2) Path competition
9. Define $a_i^j[l]$ as path $l$ with a length $i$ in stack $A$;
10. if $a_i^j[l] \neq \text{null}$ then
11. \hspace{1cm} $q_i = q_i + 1$;
12. if $q_i = Q$ then
13. \hspace{1cm} Delete all paths in $B$ with length less than or equal to $i$;
14. \hspace{1cm} end
15. \hspace{1cm} end

3) Path extension
16. Extend the paths in stack $A$ according to Algorithm 1;
17. 4) Path selection and sorting
18. After all $L$ paths in stack $A$ complete path extension;
19. if $|A| = L$, then
20. \hspace{1cm} Sort the paths in stack $B$ in descending metric;
21. else
22. \hspace{1cm} Push the $L - |A|$ paths with the maximum metrics in stack $B$ to stack $A$ and sort the rest paths in stack $B$ in descending metric;
23. \hspace{1cm} end

5) CRC-aided termination decision
25. if exist paths in $A$ with a length $N$ then
26. \hspace{1cm} Perform CRC detection;
27. \hspace{1cm} Output the path with the maximum metric as decoding sequence;
28. \hspace{1cm} else
29. \hspace{1cm} end
30. \hspace{1cm} if $q_N < Q$ then
31. \hspace{1cm} Go to step 2;
32. \hspace{1cm} else
33. \hspace{1cm} Declare a decoding failure;
34. \hspace{1cm} end
35. \hspace{1cm} end
36. \hspace{1cm} end
37. \hspace{1cm} else
38. \hspace{1cm} Go to step 2;
39. \hspace{1cm} end
40. \hspace{1cm} end

Fig. 7 Algorithm 2: LSCS($Q$, $L$, $D$) Decoder

Fig. 8 Example of the LSCS decoding with $L = 2$, $Q = 4$

4 LSCS decoding algorithm
Motivated by the advantage of SCL, we introduce list decoding to speed up the breadth search of stack decoding. A new LSCS decoding is proposed based on the path extension method in Algorithm 1 (Fig. 6). At each extension stage, LSCS chooses a maximum of $L$ paths with the largest metrics in the stack to extend simultaneously. The detailed steps are shown in Algorithm 2 (see Fig. 7) and an example is given in Fig. 8. This algorithm can offer a flexible trade-off between time complexity and computational complexity by adjusting the value of $L$. The adjustable complexity performance varies among maximum time/minimum computational complexity (similar to SCS) and minimum time/maximum computational complexity (similar to SCL) and some intermediate complexity states freely, while ensuring a stable error rate performance.

Considering the iterative property of LLR operations in (4), it is difficult to be completed synchronously, which directly affects the decoding delay and throughput. For example, log$N$ iterations are required to calculate the value of $\ell_{j}^{(1)}$. Each iteration needs to call (4). If one clock cycle (CLK) is required for each execution of (4), the calculation of $\ell_{j}^{(1)}$ requires at least log$N$ CLKs. In this manner, when $L$ paths are processed in parallel, to simplify the time complexity evaluation of LSCS decoding, we can suppose that all the parallelisable executions of (4) are performed in one CLK [25] and measure the time complexity in terms of the maximum number of CLKs for each decoding. The time complexity of LSCS decoding is given by

$$T_{\text{LSCS}} \triangleq \sum_{k=1}^{K} \left( \max_{1 \leq i \leq |A|} \ell_{j}^{(1)} \right)$$

where $\ell_{j}^{(1)}$ denotes the $j$th extending path at the $k$th extension stage. $\ell_{j}^{(1)}$ is the time complexity calculation operation and $\ell_{j}^{(1)}$ represents the number of CLKs required by the $j$th path at the $k$th extension stage, which is in the range of 1 to log$N$. $K$ denotes the total number of extension stages in the decoding process.

For the computational complexity, we refer to [22] and use the number of LLR operations in (4) to measure the decoding computational complexity, which determines the amount of energy consumption approximately. The computational complexity of LSCS decoding is expressed as

$$C_{\text{LSCS}} \triangleq \sum_{k=1}^{K} \sum_{j=1}^{L} \ell_{j}^{(1)}$$

where $\ell_{j}^{(1)}$ is the computational calculation operation. $\ell_{j}^{(1)}$ represents the number of LLR operations required by the $j$th path at the $k$th extension stage, which is in the range of 1 to log$N$.

Because the change of list size only influences the search speed and is unrelated to the search range, the error rate performance will not be affected. It is expected that with the increasing number of paths searched simultaneously at each extension stage, the search speed accelerates and LSCS will need less extension stages $K$ to find the correct path compared with SCS. Therefore, according to (14), this kind of parallelised search strategy will reduce the average decoding time complexity. However, the downside is the increase in computational complexity. Although $K$ decreases, the reduction magnitude of $K$ is not proportional to the size of $L$. This is because more incorrect paths will be searched when $L$ increases, which reduces the search efficiency. Meanwhile, $\sum_{j=1}^{L} \ell_{j}^{(1)}$ increases almost linearly with the size of $L$. The increasing number of LLR operations for an extension stage becomes the dominant influencing factor instead of the decreasing $K$. As a consequence, according to (15), the average computational complexity will increase.

When $L = 1$, the LSCS is similar to SCS, extending one path each time. In this case, the time complexity is maximised whereas the computational complexity is minimised. When $L = Q$, LSCS is similar to SCL, extending $Q$ paths with the same length simultaneously, the time complexity is minimised, whereas the computational complexity is maximised at this time. When $L$ increases from 1 to $Q$, the time complexity decreases and the computational complexity increases. Later numerical simulation results match well with the above analysis. LSCS can flexibly change its complexity performance among the two extreme states.
Define $Q$ as the maximum tolerable number of CRC check; $L$ ($1 \leq L \leq Q$) as the number of paths that are extended simultaneously at each extension stage;

1) Initialization

2) Path competition

3) Path extension 1: Extend path $u_i[\ell]$ to length $i + 1$ if $u_{i+1}$ is unfrozen bit then
   7) Reserve $(u_i[\ell], u_{i+1})$ in stack $A$ for path extension 2 directly;
   8) if $|u_{i+1}|[\ell] | \geq 0$ then
      9) Delete another path;
   else
      10) Push another path into stack $B$ for sorting process;
   end

End

4) Path competition

5) Path extension 2: Extend path $u_i[\ell + 1]$ to length $i + 2$ Extend the paths in stack $A$ according to Algorithm 1;

6) Path selection and sorting

7) CRC-aided termination decision

Fig. 9 Algorithm 3: ELSCS($Q$, $L$, $D$) Decoder

Fig. 10 Example of the ELSCS decoding with $L = 2$, $Q = 4$

and their intermediate states by adjusting the list size while the error rate performance remains unaltered.

5 Enhanced LSCS (ELCS) decoding algorithm

Conversely to SCL, where all the $L$ extending paths have the same length, in LSCS the $L$ extending paths may have different lengths if $1 < L < Q$. If the length $i$ of an extending path is odd, the number of CLKS to obtain $L_{2i+1}^{(i)}$ is only 1. Otherwise, more CLKS are required to calculate the value of $L_{2i+1}^{(i)}$. For example, it will take log $N$ CLKS to calculate the value of $L_{2i}^{(i)}$ if $i = 0$. Considering parallel processing, every extending path is assigned a processing element (PE). According to the above analysis, at the extension stage, some PEs will run for a long time while some will complete their tasks quickly and wait until other PEs stop working. This not only cannot make full use of PE resources, but also increases the decoding latency. To solve this problem, we allow each path to extend two bits at the extension stage, which means that each extension stage consists of two extensions. For example, when path $\ell$ generates new paths after the extension 1 at its first bit $i$, it does not need to wait until all other paths finish the extension 1 at their first bit, the newly generated path with a larger metric is chosen to continue to extend at the second bit $i + 1$. When all paths have finished extension 2 at their second bit, the sorting operation is performed. As a result of extending two sequential bits, the indices of the two extended bits must have an odd number and an even number, which make the runtime of PEs corresponding to different paths slightly differ from each other. This improves the PE resource utilisation efficiency. Thus, an ELSCS is proposed in Algorithm 3 (see Fig. 9) (its steps 1, 2, 4, 6, 7 are the same as Algorithm 2 (Fig. 7)) to further decrease the time complexity and a simple example is shown in Fig. 10.

Despite existing a difference between the path extension schemes of ELSCS and LSCS, the path search range of ELSCS does not diminish compared with that of LSCS due to the use of the stack. So the error rate performance will remain unchanged.

The time complexity of ELSCS decoding is given by

$$T_{ELCS} \triangleq \sum_{\ell = 1}^{l} \max_{j \in [L]} O(\ell^{(j)}) + 1$$

(16)

where $\ell^{(j)}$ denotes the $j$th extending path at extension $i$ of the $k$th extension stage. $O(\ell^{(j)})$ indicates the number of CLKS required by the $j$th extending path $\ell$ at extension $i$ of the $k$th extension stage. $\max_{i \in [L] \cup [Q]} O(\ell^{(j)})$ is the maximum number of CLKS required to extend paths with an even length. $1$ is the number of CLKS required to extend the paths with an odd length. $K$ denotes the total number of extension stages in the decoding process.

The maximum number of CLKS for an extension stage $k$ in LSCS is

$$\max_{j \in [L]} O(\ell^{(j)}) + 1$$

(17)

The maximum number of CLKS for an extension stage $k$ in ELCS is

$$\max_{j \in [L] \cup [Q]} O(\ell^{(j)}) + 1$$

(18)

Comparing (17) and (18), the average maximum number of CLKS for an extension stage in ELSCS may be slightly larger than that in LSCS. However, because of extending two bits in each extension stage, the average number of extension stages required for each decoding in ELSCS are much fewer than that in LSCS. Thus, when the average total maximum number of CLKS for a decoding process is considered, ELSCS performs better than LSCS. In other words, extension 1 reserves the extended path with a larger metric for extension 2 directly, allowing each path to extend two bits in succession without waiting for other paths, increasing the PE resource utilisation efficiency. Therefore, it will take ELSCS less time to complete the computations for a decoding process compared with LSCS.

The computational complexity of ELSCS decoding is expressed as

$$C_{ELCS} \triangleq \sum_{i = 1}^{K} \left( \sum_{j = 1}^{L} \sum_{k = 1}^{Q} O(\ell^{(j)}) \right)$$

(19)

where $O(\ell^{(j)})$ indicates the number of LLR operations required by the $j$th extending path $\ell$ at extension $i$ of the $k$th extension stage. $\sum_{i = 1}^{K} \left( \sum_{j = 1}^{L} O(\ell^{(j)}) \right)$ means the number of LLR operations for an extension stage.

For extending two sequential bits at each extension stage, compared to LSCS, the average number of LLR operations for an extension stage in ELSCS doubles while the average number of extension stages required for each decoding in ELSCS nearly halves. Thus, the average total number of LLR operations for a decoding process in ELSCS will not differ much from that in LSCS. In Section 6, simulation results also show that ELSCS has almost the same computational complexity as LSCS.

In terms of hardware resources, ELSCS and LSCS both need up to $Q$ PEs. The number is identical to that of SCL when they have the same error rate performance. However, the $Q$ PEs in LSCS/ELSCS are not necessarily all used. The exact number of used PEs is equal to list size $L$. If $L < Q$, only partial PEs are used for decoding, the remaining PEs can be used by other modules such as...
demodulation and equalisation. So LSCS/ELSCS will have a lower computing resource occupancy compared with SCL at this time. Another difference is more storage space for LSCS/ELSCS when \( L < Q \). As the algorithm is adjustable, if there is not enough memory, \( L \) can be set equal to \( Q \) in LSCS/ELSCS and the required stack depth will reduce to a minimum of \( 2L \) like in SCL, at the cost of higher computational complexity.

**6 Numerical results**

Numerical simulation results for BI-AWGNc are presented in this section to compare the block error rate (BLER) and complexity of different polar codes decoding algorithms. 10^7 code blocks are transmitted. All the used codes have code length \( N = 1024 \) and code rate \( R = 1/2 \), a CRC-24 code with generator polynomial \( g(D) = D^7 + D^6 + D^5 + D^4 + D + 1 \) is used.

Fig. 11 shows the BLER of ELSCS decoder with different \( Q \) and \( \gamma \) under different LLL threshold \( \delta \). When \( \delta \) decreases, the error rate performance degrades, especially when \( \delta < 12 \). The explanation for this behaviour is that the correct path has a higher probability of being pruned under a smaller \( \delta \), resulting in a larger decoding failure probability. It can also be observed that this trend is not significant in the low SNR regime. Because low SNR leads to small \( |E[L_i]| \), less pruning will happen and the correct path is more likely to be retained. The BLER curves of ELSCS with different list sizes all follow the same change trend as shown in Fig. 11. By comprehensive consideration of both the pruning capability and influence on BLER performance, \( \delta \) is set to 12 in the following simulation analysis. The deterioration of the error rate performance is negligible at this time and a uniform integer value threshold makes threshold comparison simple and easy for hardware implementation. In addition, we find that when the value of \( Q \) is 16, an ideal error rate performance (BLER \( \leq 2 \times 10^{-5} \)) can be obtained from \( \gamma \) greater than 2.0 dB. When \( Q = 32 \), although the error rate performance is superior, the computational complexity will be much higher as it is generally proportional to the value of \( Q \). Thus, considering the convenience of practical realisation, the value of \( Q \) is set to 16.

Fig. 12 illustrates that the error rate performance of LSCS/ELSCS remains almost unchanged with different \( L \) values and is consistent with that of SCS and SCL. Since the increasing list size only speeds up the path search and does not reduce the path search range, the probability of successful decoding does not decline. Particularly, when \( L = 16 \), the decoding principle and performance of ELSCS is similar to SCL, they both extend 16 paths with the same length each time, the difference is that SCL chooses 16 paths with the maximum metrics after sorting for the next extension stage, while, ELSCS may reserve some paths for the next extension directly without sorting operation. However, simulation results show that the performance deterioration caused by this difference can be neglected. This is because the path extension characteristics of the reserved paths match well with that of the correct one when \( \delta \) is large enough, so this path reservation strategy barely reduces the existence probability of the correct path among the decoding paths.

The BLER curve of SCL with different \( L \) at \( \gamma = 2.0 \) dB is also plotted in Fig. 12. It increases with the decrease of \( L \) and is higher than that of LSCS/ELSCS with the same \( L \). When \( L = 4 \), ELSCS can obtain a performance gain of more than 0.25 dB compared with SCL at the BLER of \( 1.4 \times 10^{-7} \) and when \( L = 2 \), the performance gain increases to 0.5 dB at the BLER of \( 7 \times 10^{-7} \).

Fig. 13 shows the normalised time complexity \( \eta_t \), and computational complexity \( \eta_c \) of different decoding algorithms with different \( L \) values. They are defined by

\[
\eta_t = \frac{T^*}{T_{\text{SCS}}} \quad \eta_c = \frac{C^*}{C_{\text{SCL}}} 
\]

where \( T^* \) and \( C^* \) indicate the average time and computational complexity of different decoding algorithms, respectively. \( T_{\text{SCS}} \) denotes the average time complexity of SCS with \( Q = 16 \) and \( C_{\text{SCL}} \) denotes the average computational complexity of SCL with \( L = 16 \). The average time and computational complexity of LSCS and ELSCS are calculated using (14)–(16), (19), respectively.

We can see that it is difficult to obtain both optimal computational and time complexity performance under certain given error performance. Minimum computational complexity often means maximum time complexity and vice versa. For LSCS and ELSCS, we can adjust the trade-off between computational and time complexity by changing the list size. In Fig. 13, the BLER is constant at \( 2 \times 10^{-5} \). When \( L = 1 \), the decoding principle and performance of ELSCS are similar to SCS, achieving the minimum computational complexity and the maximum time complexity. On the contrary, when \( L = 16 \), ELSCS extends 16 paths with the same length each time, similar to SCL, thus, has the maximum computational complexity and the minimum time complexity. When the list size is between 1 and 16, there will be many intermediate performance states. As list size increases, the average maximum number of CLKs for each extension stage may become slightly larger. However, the average number of path extension stages required for each decoding drops, leading to the decline of the average total maximum number of CLKs for each decoding. Consequently, the time complexity decreases. When considering the computational complexity, the average number of LLR operations for each extension stage increases dramatically with increasing list size and becomes the dominant influencing factor.
Hence, the average total number of LLR operations rises and the computational complexity increases. As a comparison, ELSCS without LTPE scheme is depicted. It can be found that LTPE scheme reduces both time and computational complexity. This is because fewer candidate partial decoding paths are inserted into the stack as a result of the path pruning capability of LTPE scheme. The decoder does not need to search among many paths that are improbable to become the final decoding output. Consequently, ELSCS with LTPE scheme can get lower decoding time and computations compared to ELSCS without LTPE scheme. Also, ELSCS without LTPE scheme can still provide a flexible trade-off between time complexity and computational complexity. The complexity performance of two extreme cases when $L = 1$ and $L = 16$ is the same as that of SCS and SCL, respectively. Thus, even without LTPE scheme, ELSCS is still able to provide a low time complexity or computational complexity through the adjustment of list size $L$.

We can also observe that the computational complexity curves of LSCS and ELSCS overlap but the time complexity of ELSCS reduces in comparison to that of LSCS. As ELSCS extends two sequential bits at each path extension stage, the PE resource utilisation efficiency is improved and the time complexity has a significant reduction. When $L = 8$, the relative time complexity of ELSCS is reduced by 20.42% compared with that of LSCS.

As shown in Fig. 14, we can consider choosing $L = 1, 2, 4, 8, 12, 16$ as different modes of ELSCS algorithm, thereby providing an adjustable decoding algorithm for different cases. A proper configuration of $L$ can make the algorithm meet the specific decoding latency and throughput demand, while, reducing the computational complexity and the occupied number of PEs to the least possible degree. As a comparison, although SCL ($L = 16$) can provide smaller latency, it requires more computational complexity and occupies more computing resources. When the decoding latency of ELSCS already meets the throughput requirement of the application, a smaller decoding latency with the higher cost is not necessary. In the practical application of ELSCS algorithm, the value of $Q$ is chosen firstly, which determines the error rate performance. Generally, $Q$ is set to 16. Then the value of $L$ is adjusted to select the appropriate algorithm mode, ensuring that the required throughput is achieved with the lowest computational complexity and minimum computing resources.

Figs. 15 and 16 show the statistical results of the maximum stack depth and the average stack depth required by SCS and ELSCS. The stack depth is 1000 and represents the total depth of stack $\mathcal{A}$ and stack $\mathcal{B}$. We observe that LTPE scheme reduces the required storage size dramatically. Compared with SCS, the maximum stack depth drops 70%, from 1000 to around 300, which means that a stack with a depth of 300 is enough for the proposed algorithm. At $\gamma = [1.5, 2.0, 2.5, 3.0 \text{ dB}]$, the average storage space of ELSCS reduces to 22.27, 15.92, 6.57, 1.19% of that of SCS, respectively. According to the analysis in Section 3, the value of $E[L_i]\gamma$ at unfrozen bit will become larger when $\gamma$ increases. Thus more paths will be pruned because of a large value of $|L_i|\gamma$. This explains why both the maximum and average depths become smaller when $\gamma$ increases. The simulation results corroborate the effectiveness of LTPE scheme in path pruning. We can see that a large amount of storage space can be optimised by using ELSCS decoder for polar codes instead of conventional stack decoder.

7 Conclusion

In this paper, we have proposed a complexity-adjustable multimode decoding algorithm ELSCS for polar codes. We first study the LLR characteristics of the correct path in the decoding process. It was observed that if the decoding bit of the correct path is an unfrozen bit, it happens with a large probability that the path metric difference between two generated paths after the path extension is large. An LTPE scheme is designed using this fact to reduce the storage space. Based on the proposed scheme, we employ both the ideas of SCL and SCS to introduce a novel LSCS algorithm, which can combine their complementary advantages. By changing the list size of extending paths, LSCS can adjust the trade-off between time complexity and computational complexity flexibly while retaining the error performance unchanged. Driven by the low PE utilisation problem in LSCS, LSCS is improved to obtain an enhanced version, ELSCS. ELSCS extends two sequential bits at extension stage to make the runtime of different PEs close and reduce the occurrences of some PEs staying idle and waiting. Thus, the time complexity can further decrease.
References

This work was supported in part by the National Natural Science Foundation of China under Grant 61701155.

9 References