Combating Error Propagation in Window Decoding of Braided Convolutional Codes

Min Zhu*, David G. M. Mitchell†, Michael Lentmaier‡, Daniel J. Costello, Jr.§, and Baoming Bai*
*State Key Laboratory of ISN, Xidian University, Xi'an, P. R. China, mzhu@xidian.edu.cn, bmbai@mail.xidian.edu.cn
†Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM, USA, dgmm@nmsu.edu
‡Department of Electrical and Information Technology, Lund University, Lund, Sweden, michael.lentmaier@eit.lth.se
§Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN, USA, dcostel1@nd.edu

***Abstract*—In this paper, we study sliding window decoding of braided convolutional codes (BCCs) in the context of a streaming application, where decoder error propagation can be a serious problem. A window extension algorithm and a resynchronization mechanism are introduced to mitigate the effect of error propagation. In addition, we introduce a soft bit-error-rate stopping rule to reduce computational complexity, and the tradeoff between performance and complexity is examined. Simulation results show that, using the proposed window extension algorithm and resynchronization mechanism, the error performance of BCCs can be improved by up to three orders of magnitude with reduced computational complexity.**

I. Introduction

Braided convolutional codes, first introduced in [1], are a counterpart to braided block codes (BBCs) [2] which can be regarded as a diagonalized version of product codes [3] or expander codes [4]. In contrast to BBCs, BCCs use short constraint length convolutional codes as component codes. The encoding of BCCs can be described by a two-dimensional sliding array, where each symbol is protected by two component convolutional codes. BCCs are a type of parallel-concatenated convolutional code in which the parity outputs of one component encoder are fed back and used as inputs to the other component encoder at the succeeding time unit. Two variants of BCCs were considered in [1]. Tightly braided convolutional codes (TBCCs) are obtained if a dense array is used to store the information and parity symbols. This construction is deterministic and simple but performs relatively poorly due to the absence of randomness. Alternatively, sparsely braided convolutional codes (SBCCs) that employ random permutors have low density, resulting in improved iterative decoding performance [1]. Moloudi *et al*. considered SBCCs as spatially coupled turbo-like codes and showed that threshold saturation occurs for SBCCs over the binary erasure channel [5], [6]. SBCCs can operate in bitwise or blockwise modes, according to whether convolutional or block permutors are employed. It was also shown numerically that the free (minimum) distance of bitwise (blockwise) SBCCs grows linearly with the overall constraint length, leading to the conjecture that SBCCs are asymptotically good [1], [6].

Due to their turbo-like structure, BCCs can be decoded with iterative decoding. Analogous to LDPC convolutional codes [7], [8], SBCCs can employ sliding window decoding for low latency operation. Unlike window decoding of LDPC convolutional codes, which typically uses an iterative belief-propagation (BP) message passing algorithm, window decoding of SBCCs is based on the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm. It has been shown that blockwise SBCCs with sliding window decoding have capacity-approaching performance [9], but for large frame lengths or streaming applications, SBCCs are susceptible to severe but infrequent decoder error propagation. That is, once a block decoding error occurs, decoding of the following blocks can be affected, which can in turn cause a continuous string of block errors and result in unacceptable performance loss.

In this paper, we study several error propagation mitigation techniques for SBCCs. Specifically, a window extension algorithm and a resynchronization mechanism are introduced to combat error propagation. In addition, a soft bit-error-rate (BER) stopping rule is proposed to reduce decoding complexity and, the resulting tradeoff between decoding performance and decoding complexity is explored.

II. Continuous Transmission of Braided Convolutional Codes

In this section, we briefly review continuous encoding and sliding window decoding of blockwise SBCCs. For details, please refer to [1] and [9].

A. Continuous Encoding

Sparsely braided convolutional codes are constructed using an infinite two-dimensional array consisting of one horizontal and one vertical encoder. These two encoders are linked through parity feedback. In this manner, the systematic and parity symbols are "braided" together. In this paper, we limit ourselves to rate $R = 1/3$ blockwise SBCCs as an example, but generalization to other rates is straightforward. In this case, the information sequence enters the encoder in a block-by-block manner, typically with a relatively large block size. Fig. 1 is a conceptual illustration of the continuous encoding process for a rate $R = 1/3$ blockwise SBCC, which utilizes two recursive systematic convolutional (RSC) component encoders each of rate $R_{cc} = 2/3$, where $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ are each block permutors of length T. The information sequence is divided into blocks of length T symbols, i.e.,

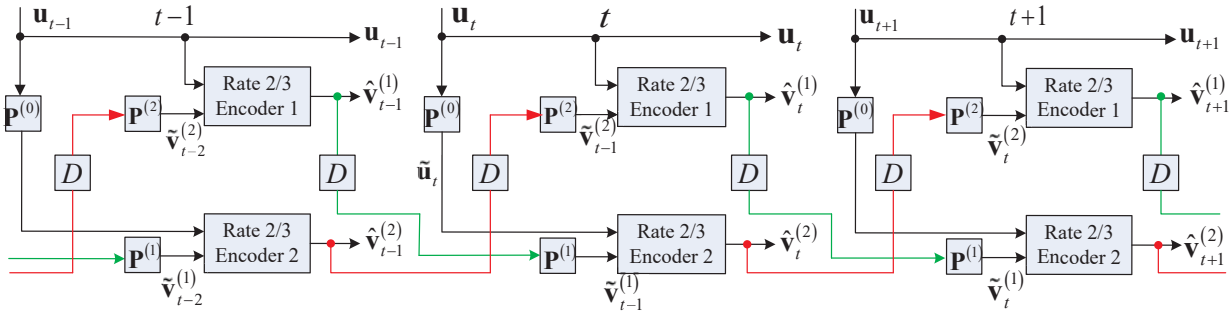


Fig. 1. Continuous encoder chain for a rate $R = 1/3$ blockwise SBCC.

$\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t, \ldots)$, where $\mathbf{u}_t = (u_{t,1}, u_{t,2}, \ldots, u_{t,T})$. At time t, $\mathbf{u}_t$ is interleaved using $\mathbf{P}^{(0)}$ to form $\tilde{\mathbf{u}}_t$, and $\mathbf{u}_t$ and $\tilde{\mathbf{u}}_t$ enter the component encoders. The parity outputs $\hat{\mathbf{v}}_t^{(i)}$, $i \in \{1, 2\}$, at time t are delayed by one time unit, interleaved using $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$, respectively, and then enter the component encoders as the input sequences $\tilde{\mathbf{v}}_t^{(i)}$, $i \in \{1, 2\}$, at time $t + 1$. The information sequence $\mathbf{u}$, the parity output sequence $\hat{\mathbf{v}}_t^{(1)}$ of encoder 1, and the parity output sequence $\hat{\mathbf{v}}_t^{(2)}$ of encoder 2 are sent over the channel. For initialization, at time instant 0, we assume that $\tilde{\mathbf{v}}_{-1}^{(1)} = \mathbf{0}$ and $\tilde{\mathbf{v}}_{-1}^{(2)} = \mathbf{0}$.

Transmission can be terminated after a frame consisting of L blocks, resulting in a slight rate loss, or proceed in an unterminated (streaming) fashion, in which case the rate is given by $R = \frac{1}{3}$.

B. Sliding Window Decoding

In order to help describe the proposed error propagation mitigation methods, the structure of the sliding window decoder [9] is shown in Fig. 2. The *window size* is denoted as w. The block at time instant t is the *target block* for decoding in the window containing the blocks received at times t to $t+w-1$. Briefly, the decoding process in a window beings with I_1 turbo, or *vertical*, iterations on the target block at time t, during which the two component convolutional codes pass soft messages on the T information bits in that block to each other. Then, soft messages on the parity bits are passed forward, and I_1 vertical iterations are performed on the block at time $t+1$. This continues until I_1 vertical iterations are performed on the last received block in the window. Then the process is repeated in the backward direction (from the last block to the first block in the window) with soft messages being passed back through the $2w$ BCJR decoders. This round trip of decoding is called a *horizontal* iteration. After I_2 horizontal iterations, the T target symbols are decoded, and the window shifts forward to the next position, where the T symbols at time $t + 1$ become the target symbols.

C. Error Propagation

Since an encoded block in a blockwise BCC affects the encoding of the next block (see Fig. 1), each time a block of target symbols is decoded, the log-likelihood ratios (LLRs) associated with the decoded symbols also affect the decoding of the next block. Hence, if, after the maximum number of decoding iterations, some unreliable LLRs remain in the target block, causing a block decoding error, those unreliable LLRs can potentially trigger a string of additional block errors, resulting in *error propagation*. To illustrate this effect, we consider two identical 4-state RSC component encoders whose generator matrix is given by

$$G(D) = \begin{pmatrix} 1 & 0 & \frac{1}{1+D+D^2} \\ 0 & 1 & \frac{1+D^2}{1+D+D^2} \end{pmatrix}, \tag{1}$$

where we assume the encoders are left unterminated at the end of each block. The three block permutors $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ were chosen randomly with the same size $T = 8000$. We assume that transmission stops after a frame of L blocks is decoded and a uniform decoding schedule (see [9] for details) is used. The bit error rate (BER), block error rate (BLER), and frame error rate (FER) performance for transmission over the AWGN channel with BPSK signalling is given in Fig. 3, where the window size $w = 3$, the number of vertical iterations is $I_1 = 1$, the number of horizontal iteration is $I_2 = 20$, and the frame length is $L = 1002$ blocks.

From Fig. 3, we see that the rate $R = 1/3$ blockwise SBCC performs about 0.6 dB away from the Shannon limit. Even so, among the 10000 simulated frames, several were observed to exhibit error propagation. For example, 9 such frames were observed at $E_b/N_0 = 0.04$. In order to depict the error propagation phenomenon clearly, we give the bit error distribution per block of one frame with error propagation in Fig. 4(a), for $E_b/N_0 = 0.04$ dB. We see that, for $I_2 = 20$, from the 830th block on, the number of error bits is large, and the errors continue to the end of the frame, a clear case of error propagation. For $I_2 = 30$, error propagation starts two blocks later than for $I_2 = 20$, but the overall effect of increasing the number of iterations is minimal. On the other hand the bit error distribution per block, based on 10000 simulated frames with two different window sizes, is shown in Fig. 4(b), where we see that increasing the window size from 3 to 4 reduces the number of error propagation frames from 9 to 1, thus significantly improving performance.

For larger frame lengths, and particularly for streaming transmission, error propagation will severely degrade the decoding performance illustrated in Fig. 3. Hence, we now introduce two ways of mitigating the error propagation effect in sliding window decoding of SBCCs.

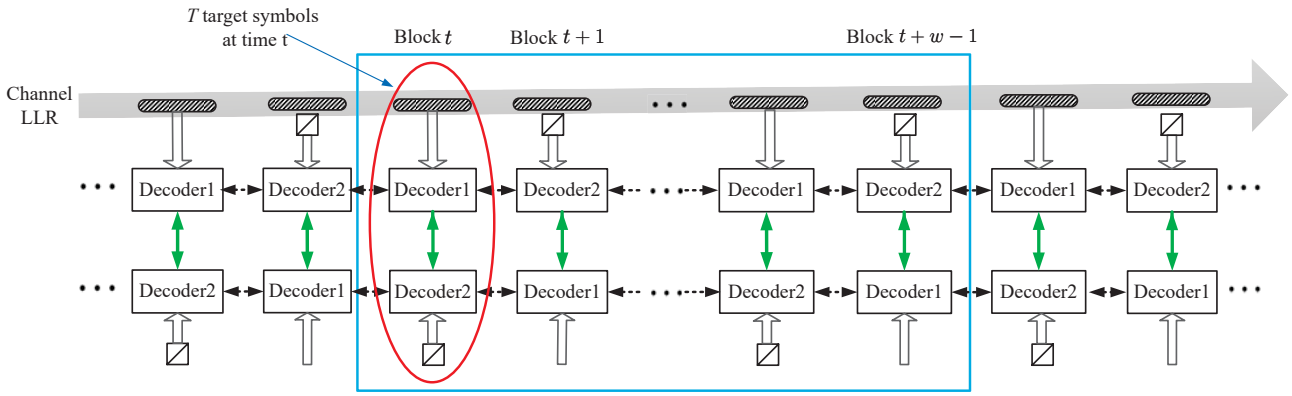


Fig. 2. Continuous sliding window decoder for blockwise SBCCs [9].

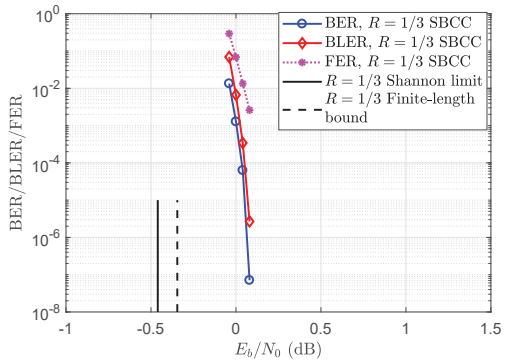
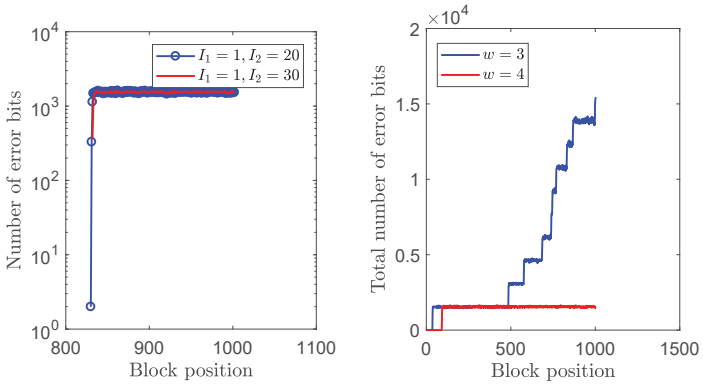


Fig. 3. The BER, BLER, and FER performance of rate $R = 1/3$ SBCCs.



(a) (b)

Fig. 4. The error distribution per block for rate $R = 1/3$ blockwise SBCCs: (a) One frame with different numbers of iterations, $w = 3$, and (b) 10000 frames with different window sizes, $I_1 = 1$, $I_1 = 20$.

III. Error Propagation Mitigation

In this section, we propose a window extension algorithm and a resychronization mechanism to mitigate the effect of error propagation in SBCCs.

A. Window Extension Algorithm

In [9], window decoding of SBCCs is performed with a fixed window size. Based on the results presented in Fig. 4(b), we introduce a variable window size concept for sliding window decoding. Before describing the window extension algorithm, we give some definitions. Let $\mathbf{L}_d^{(i,j)} = \{l_{d,0}^{(i,j)}, l_{d,1}^{(i,j)}, l_{d,2}^{(i,j)}, \ldots, l_{d,T-1}^{(i,j)}\}$ denote the decision LLRs of the T information bits in the ith block of the current window after the jth horizontal iteration. Then the average absolute LLR of the T information bits after the jth horizontal iteration is given by $\bar{L}_d^{(i,j)} = \frac{1}{T}\sum_{k=0}^{T-1}\left|l_{d,k}^{(i,j)}\right|$.

During the decoding process, the window extension algorithm operates as follows: when the number of horizontal iterations reaches its maximum value I_2, if any of the average absolute LLRs of the first τ blocks in the current window, $1 \le \tau \le w$, is lower than a predefined threshold θ, that is, if

$$\bar{L}_d^{(i,j)} < \theta, \qquad i = 0, 1, \ldots, \tau - 1, \tag{2}$$

the target block is not decoded, the window size is increased by 1, and the decoding process restarts with horizontal iteration number 0. This process continues until either the target block is decoded or the window size reaches a predefined maximum $w_{\max}$, in which case the target block is decoded regardless of whether (2) is satisfied. Assuming an initial window size $w = 3$, Fig. 5 illustrates how the decoder window size increases by 1 each time (2) is met, up to a maximum window size of $w_{\max} = 6$.[1] Full details of the window extension algorithm are given in an expanded version of the paper posted online [10].

For the same simulation conditions used in Fig. 3, the BER, BLER, and FER performance of rate $R = 1/3$ blockwise SBCCs with the window extension algorithm is shown in Fig. 6, where $w_{\max} = 6$, $\tau = 2$, and $\theta = 10$. We see that rate $R = 1/3$ blockwise SBCCs with window extension show an order of magnitude improvement in BER, BLER, and FER compared to the results of Fig. 3. We also remark that, even though $w_{\max} = 6$, the average window size $\bar{w}$ is found to be only slightly larger than w, e.g. $\bar{w} = 3.0014$ for $E_b/N_0 = 0.04$ dB, since window extension is only activated in the few cases when error propagation is detected.

B. Resynchronization Mechanism

We see from Fig. 6 that the window extension algorithm greatly reduces the effect of error propagation. However, for

[1] We can use the existing hardware serially, with additional memory, and set $w_{\max}$ as high as needed, but $w_{\max} = 6$ gives a reasonable tradeoff among complexity, memory requirements, and delay.

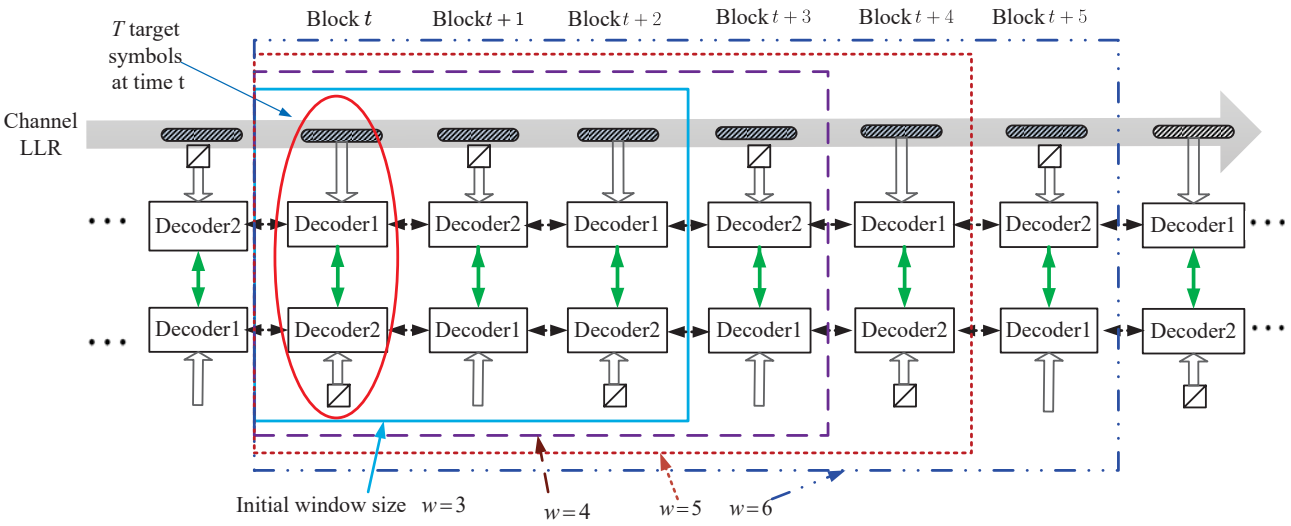


Fig. 5. Decoder with the window extension algorithm.

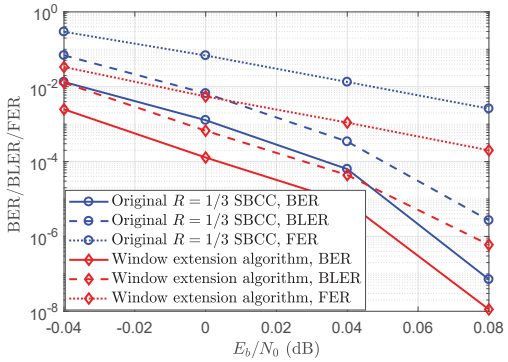


Fig. 6. BER/BLER/FER performance comparison of rate $R=1/3$ blockwise SBCCs with and without the window extension algorithm.

very long frames or for streaming, even one occurrence of error propagation can be catastrophic. We now introduce a resynchronization mechanism to address this problem.[2]

As noted above, the first block in a BCC encoder chain has two known input sequences. Therefore, the input LLRs in the first block are more reliable than for the succeeding blocks. Motivated by this observation, and assuming the availability of a noiseless binary feedback channel, we propose that, when the window extension algorithm is unable to stop error propagation, the encoder resets to the initial 0 state and begins the encoding of a new chain. This resynchronization mechanism is described below.

After a target block is decoded in the window extension algorithm, if its average absolute LLRs satisfy $\bar{L}_d^{(0,I_2)} < \theta$, we consider this target block as *failed*. If we experience N_r consecutive failed target blocks, we declare an error propagation condition and initiate encoder and decoder resychronization using the feedback channel. In other words, the encoder 1) sets the initial register states of the two component convolutional encoders to "0", and 2) begins encoding the next block with two known (all "0") input sequences together with the new information block. Meanwhile, the decoder makes decisions based on the current LLRs for the remaining blocks in the current window and restarts decoding once w new blocks are received. Full details of the resynchronization algorithm are given in an expanded version of the paper posted online [10].

To demonstrate the efficiency of the resynchronization mechanism alongside the extension technique, the BER, BLER, and FER performance of rate $R=1/3$ blockwise SBCCs with the window extension algorithm and the resynchronization mechanism is shown in Fig. 7 for the same simulation conditions used in Fig. 6 and $N_r=1$. We see that, compared to the results of Fig. 3, rate $R=1/3$ blockwise SBCCs with window extension and resynchronization gain approximately three orders of magnitude in BER and BLER and about one order of magnitude in FER at low SNRs.[3] (At high SNRs, the curves tend to merge, since error propagation, and thus the need for window extension and resynchronization, is very rare under good channel conditions.)

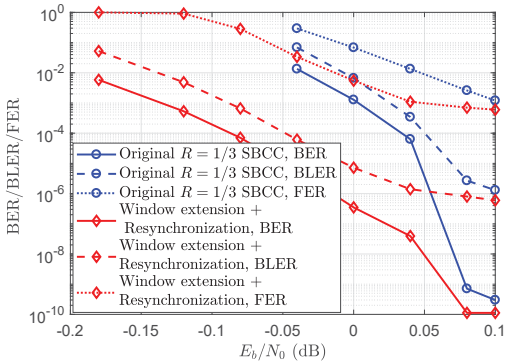


Fig. 7. BER/BLER/FER comparison of rate $R=1/3$ blockwise SBCCs with and without window extension and resynchronization.

[2]Resynchronization could be employed with or without window extension but, due to the low cost and effectiveness of extension, we assume the techniques are used together.

[3]Although the resynchronization mechanism terminates error propagation in a frame, thus improving both BER and BLER, it does not further reduce the number of frames in error.

IV. Early Stopping Rule

The decoding complexity of BCCs with sliding window decoding depends mainly on the number of horizontal iterations. Therefore, in order to minimize unnecessary horizontal iterations, we introduce a soft BER stopping rule, which was first proposed for LDPC convolutional codes in [11]. Every time a horizontal iteration finishes, the average estimated BER BER_{est} of the target bits in the current window is obtained using the following steps:

- Calculate the decision LLR (the sum of the channel LLR, the prior LLR, and the extrinsic LLR) L_d^j of every information bit in the target block, $j = 0, 1, \ldots, T-1$;
- Compute the average estimated BER of the target information bits is

$$BER_{est} = \frac{1}{T}\sum_{j=0}^{T-1} 1.0/\left(1.0 + \exp\left(\left|L_d^j\right|\right)\right).$$

If the average estimated BER of the target bits satisfies $BER_{est} \leq \gamma$, decoding is stopped and a decision on the target symbols in the current window is made.

Note that the window extension algorithm, the resynchronization mechanism, and the soft BER stopping rule can operate together in a sliding window decoder. We now give an example to illustrate the tradeoffs between performance and computational complexity when error propagation mitigation is combined with the stopping rule. Fig. 8 shows the performance of rate $R = 1/3$ blockwise SBCCs with window extension, resynchronization, and the stopping rule for the same simulation conditions used in Fig. 7 and $\gamma = 10^{-7}$. We see that using the stopping rule degrades the BER performance only slightly, but the BLER performance is negatively affected in the high SNR region.[4] The average number of horizontal iterations per block is also shown in Fig. 9, where we see that the stopping rule greatly reduces the number of horizontal iterations, especially in the high SNR region.

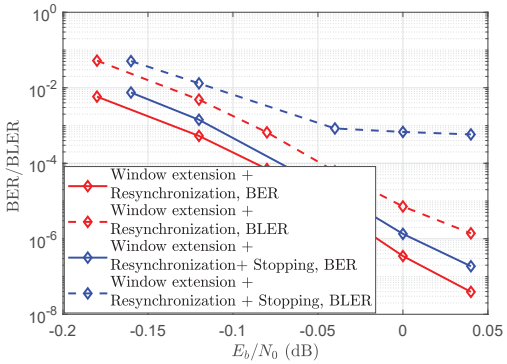


Fig. 8. BER/BLER comparison of rate $R = 1/3$ blockwise SBCCs with window extension and resynchronization, with and without the stopping rule.

V. Conclusion

In this paper we investigated the severe but infrequent error propagation problem associated with blockwise SBCCs. A window extension algorithm and a resynchronization mechanism were proposed to mitigate error propagation, which can have a catastrophic effect on the performance for large frame lengths and continuous streaming operation. The BER and BLER performance of blockwise SBCCs with these two mitigation methods was shown to outperform the original blockwise SBCCs by about three orders of magnitude. Furthermore, a soft BER stopping rule was introduced and shown to significantly reduce decoding complexity with little effect on BER performance.

[4]The BLER loss at high SNR can be reduced by using a smaller γ at a cost of some increased complexity.

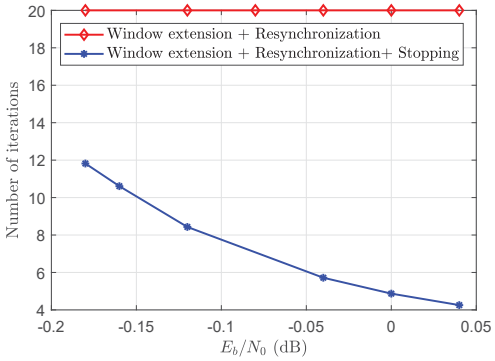


Fig. 9. Number of horizontal iterations of rate $R = 1/3$ blockwise SBCCs with window extension and resynchronization, with and without the stopping rule.

Acknowledgment

This work was supported in part by U. S. NSF Grant CCSS-1710920, by NSFC Grant 61701368, and by NSFC Grant 61771364.

References

[1] W. Zhang, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Braided convolutional codes: a new class of turbo-like codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 316-331, Jan. 2010.

[2] A. J. Feltström, M. Lentmaier, D. V. Truhachev, and K. S. Zigangirov, "Braided block codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2640-2658, Jun. 2009.

[3] P. Elias, "Error free coding," *IRE Trans. Inf. Theory*, vol. 4, no. 4, pp. 29-37, Sep. 1954.

[4] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710-1722, Nov. 1996.

[5] S. Moloudi, M. Lentmaier, and A. Graell i Amat, "Spatilly coupled turbo-like codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 10, pp. 6199-6215, 2017.

[6] S. Moloudi, M. Lentmaier, and A. Graell i Amat, "Finite length weight enumerator analysis of braided convolutional codes," in *Proc. Int. Symp. Inf. Theory and Its Applications*, Monterey, CA, USA, Oct. 2016, pp. 488-492.

[7] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K. S. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5274-5289, Oct. 2010.

[8] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, "Windowed decoding of protograph-based LDPC convolutional codes over erasure channels," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2303-2320, April 2012.

[9] M. Zhu, D. G. M. Mitchell, M. Lentmaier, D. J. Costello, Jr., and B. Bai, "Braided convolutional codes with sliding window decoding," *IEEE Trans. on Communications*, vol. 65, no. 9, pp. 3645-3658, Sept. 2017.

[10] M. Zhu, D. G. M. Mitchell, M. Lentmaier, D. J. Costello, Jr., and B. Bai,"Combating error propagation in window decoding of braided convolutional codes," available at *http://arxiv.org/abs/1801.03235*.

[11] N. Ul Hassan, A E. Pusane, M. Lentmaier, G. P. Fettweis, and D. J. Costello, Jr., "Non-uniform window decoding schedules for spatially coupled LDPC codes," *IEEE Trans. on Communications*, vol. 65, no. 2, pp. 501-510, Nov. 2016.