Braided Convolutional Codes with Sliding Window Decoding

Zhu, Min; Mitchell, David G. M.; Lentmaier, Michael; Costello, Daniel J. Jr.; Bai, Baoming

Published in:
IEEE Transactions on Communications

DOI:
10.1109/TCOMM.2017.2707073

2017

Document Version:
Peer reviewed version (aka post-print)

Braided Convolutional Codes with Sliding Window Decoding

Min Zhu, *Member, IEEE*, David G. M. Mitchell, *Senior Member, IEEE*, Michael Lentmaier, *Senior Member, IEEE*, Daniel J. Costello, Jr., *Life Fellow, IEEE*, and Baoming Bai, *Member, IEEE*

***Abstract*—In this paper, we present a novel sliding window decoding scheme based on iterative BCJR decoding for braided convolutional codes, a class of turbo-like codes with short constraint length component convolutional codes. The tradeoff between performance and decoding latency is examined and, to reduce decoding complexity, both uniform and nonuniform message passing schedules within the decoding window, along with early stopping rules, are proposed. We also perform a density evolution analysis of sliding window decoding to guide the selection of the window size and message passing schedule. Periodic puncturing is employed to obtain rate-compatible code rates of 1/2 and 2/3 starting from a rate 1/3 mother code and a code rate of 3/4 starting from a rate 1/2 mother code. Simulation results show that, with nonuniform message passing and periodic puncturing, near capacity performance can be maintained throughout a wide range of rates with reasonable decoding complexity and no visible error floors.**

***Index Terms*—Braided convolutional codes, sliding window decoding, iterative decoding, turbo-like codes, decoding latency.**

I. Introduction

Braided block codes (BBCs) [1] can be regarded as a diagonalized version of product codes [2] or expander codes [3], [4]. A BBC is constructed by interconnecting two block component codes such that information symbols are checked by both component encoders, and the parity symbols of one component encoder are used as inputs to the other component encoder. Recently, BBCs with Bose-Chaudhuri-Hocqenghem (BCH) component codes [5] and the closely related staircase codes [6] have been investigated for high-speed optical communication and have been found to achieve excellent performance with iterative hard decision decoding.

As a counterpart of BBCs, braided convolutional codes (BCCs) [7], a class of turbo-like codes that can be decoded with iterative decoding based on the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm, were first introduced in [8]; however, in contrast to BBCs, BCCs use short constraint length convolutional codes as component codes. BBCs and BCCs are similar in terms of the encoding process. The encoding of BCCs can be described by a two-dimensional sliding array, where each information symbol is protected by two component convolutional codes. The connections between the two component encoders are defined by the positions where information symbols and parity symbols are stored in the two-dimensional array. Analogous to BBCs, a tightly braided convolutional (TBC) code results when a dense array is used to store the information and parity symbols. Alternatively, sparsely braided convolutional (SBC) codes have low density, resulting in improved iterative decoding performance [7]. It was also shown (numerically) in [7] that the minimum distance of SBC codes grows linearly with the overall constraint length, leading to the conjecture that SBC codes are asymptotically good. In [9]–[11], the threshold (with belief propagation (BP) decoding) of SBC codes was analyzed on the binary erasure channel, and it was demonstrated that threshold saturation occurs, i.e., SBC codes behave in a manner similar to LDPC convolutional (spatially coupled) codes.

In this paper, we build on our work in [12] and introduce a new decoding scheme for SBC codes. Instead of the pipeline decoder used in [7] and [8], a sliding window decoder is proposed for SBC codes operating over the binary-input additive white Gaussian noise (AWGN) channel.[1] Window decoding, which has been extensively studied for LDPC convolutional codes [13]–[15], provides a simple and efficient way to trade off decoding performance for reduced latency and memory requirements. Unlike window decoding of LDPC convolutional codes, which typically uses an iterative BP message passing algorithm based on belief propagation, window decoding of SBC codes is based on the BCJR algorithm. In addition, the tradeoff between performance and decoding latency is explored, and we discuss how to choose the code parameters and the decoder window size to achieve the best performance when the decoding latency is fixed. The computational complexity of window decoding for BCCs is then analysed. In order to reduce the decoding complexity, different message passing schedules (both uniform and nonuniform [16]) between the component BCJR decoders are investigated. Also, to further reduce the computational complexity, we propose two stopping

This work was presented in part at the IEEE Information Theory Workshop, Jeju Island, Korea, October 2015, and at the Information Theory and Applications Workshop, San Diego, CA, February 2016.

M. Zhu and B. Bai are with the State Key Lab. of ISN, Xidian University, Xi'an 710071, China, (e-mail: zhunanzhumin@gmail.com; bmbai@mail.xidian.edu.cn).

D. G. M. Mitchell is with the Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM 88003, USA, (e-mail: dgmm@nmsu.edu).

D. J. Costello, Jr. is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA, (e-mail: dcostel1@nd.edu).

M. Lentmaier is with the Department of Electrical and Information Technology, Lund University, Sweden. (e-mail: michael.lentmaier@eit.lth.se).

This work was supported in part by the U. S. NSF under Grant CCF-1161754, by the NSFC under Grant 61372074, by the 973 Program of China under Grant 2012CB316100, and by the Joint Ph.D. Fellowship Program of the China Scholarship Council.

[1] Although we focus on the binary-input AWGN channel in this paper, the proposed sliding window decoder for SBC codes is not limited to this case.

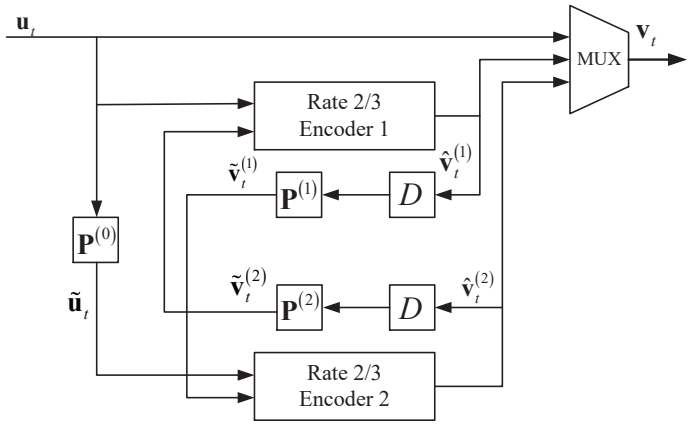


Fig. 1. Encoder for a rate $R = 1/3$ blockwise SBC code.

rules: one based on a cross entropy [17] measure which was originally designed as a stopping rule for turbo codes; the other based on the magnitude of the reliability measure produced by the BCJR algorithm. Finally, we consider periodic puncturing of SBC codes to achieve rate-compatible SBC codes.

II. Sparsely Braided Convolutional Codes

Sparsely braided convolutional (SBC) codes are constructed using an infinite two-dimensional array and consist of two recursive systematic convolutional (RSC) encoders linked through parity feedback. In this manner, the information and parity symbols are "braided" together. There are two types of SBC codes: bitwise and blockwise. Bitwise uses convolutional interleavers and transmission is continuous, while blockwise uses block interleavers and transmission occurs in finite-length blocks. In this paper we use rate $R = 1/3$ blockwise SBC codes for illustration, although extensions to other rates are straightforward. An example of a rate $R = 1/3$ blockwise SBC encoder is shown in Fig. 1. It consists of two RSC component encoders each of rate $R_{cc} = 2/3$. The information sequence is divided into blocks of T symbols, i.e., $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t, \ldots)$, where $\mathbf{u}_t = (u_{t,1}, u_{t,2}, \ldots, u_{t,T})$, and $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ are each block permutors of length T.

At time unit $t = 0$, information block $\mathbf{u}_0$ and its permuted version $\tilde{\mathbf{u}}_0 = \mathbf{u}_0\mathbf{P}^{(0)}$ enter the first inputs of Encoder 1 and Encoder 2, respectively, one bit at a time. Meanwhile, blocks $\tilde{\mathbf{v}}_{-1}^{(2)}$ and $\tilde{\mathbf{v}}_{-1}^{(1)}$, consisting of T zeros each (the initial condition), enter the second inputs of Encoder 1 and Encoder 2, respectively, also one bit at a time. Encoders 1 and 2 then generate length T parity blocks $\hat{\mathbf{v}}_0^{(i)} = \left(\hat{v}_{0,1}^{(i)}, \hat{v}_{0,2}^{(i)}, \cdots, \hat{v}_{0,T}^{(i)}\right)$, $i = 1, 2$, and blocks $\mathbf{v}_0^{(0)} = \left(v_{0,1}^{(0)}, v_{0,2}^{(0)}, \cdots, v_{0,T}^{(0)}\right) \triangleq \mathbf{u}_0$, $\mathbf{v}_0^{(1)} \triangleq \hat{\mathbf{v}}_0^{(1)}$, and $\mathbf{v}_0^{(2)} \triangleq \hat{\mathbf{v}}_0^{(2)}$ are multiplexed and sent over the channel. In general, at time unit t, parity block $\mathbf{v}_t^{(1)}$ is calculated by Encoder 1 as a function of $\mathbf{u_t}$ and $\tilde{\mathbf{v}}_t^{(2)} = \mathbf{v}_{t-1}^{(2)}\mathbf{P}^{(2)}$. Similarly, parity block $\mathbf{v}_t^{(2)}$ is calculated by Encoder 2 as a function of $\tilde{\mathbf{u}}_t = \mathbf{u}_t\mathbf{P}^{(0)}$ and $\tilde{\mathbf{v}}_t^{(1)} = \mathbf{v}_{t-1}^{(1)}\mathbf{P}^{(1)}$. Note that there is a one time unit delay prior to permutors $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$. The blocks $\mathbf{v}_t^{(0)} = \left(v_{t,1}^{(0)}, v_{t,2}^{(0)}, \cdots, v_{t,T}^{(0)}\right) \triangleq \mathbf{u}_t$, $\mathbf{v}_t^{(1)} = \hat{\mathbf{v}}_t^{(1)}$, and $\mathbf{v}_t^{(2)} = \hat{\mathbf{v}}_t^{(2)}$ are then multiplexed into the code sequence $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where

$$\mathbf{v}_t = \left(v_{t,1}^{(0)}, v_{t,1}^{(1)}, v_{t,1}^{(2)}, v_{t,2}^{(0)}, v_{t,2}^{(1)}, v_{t,2}^{(2)}, \ldots, v_{t,T}^{(0)}, v_{t,T}^{(1)}, v_{t,T}^{(2)}\right), \tag{1}$$

is a $3T$-bit vector, and sent over the channel.

At the end of transmission, termination of the overall code is used to protect the final information blocks. In this case, after the LT-bit information sequence $\mathbf{u}_{[0,L-1]} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{L-1})$ enters the blockwise SBC encoder, Λ additional all-zero blocks $\mathbf{u}_L, \ldots, \mathbf{u}_{L+\Lambda-1}$ enter the encoder. These Λ all-zero blocks are not sent over the channel but the resulting parity blocks are transmitted. The *actual rate* of the SBC code, including the tail, is thus given by

$$\tilde{R} = \frac{LT}{3LT + 2\Lambda T} = \frac{1}{3}\frac{L}{L + (2\Lambda/3)}, \tag{2}$$

and we see that, for fixed Λ, the actual rate $\tilde{R}$ of the SBC code approaches $R = 1/3$ as $L \to \infty$. A terminated sequence of $L + \Lambda$ blocks will be referred to as a *frame*. In this paper, we consider the rate $R_{cc} = 2/3$ RSC component encoders to be unterminated, i.e., the initial encoder states at time unit $t + 1$ are the same as the final encoder states at time unit t.[2]

III. Sliding Window Decoding

A parallel pipeline decoding architecture was proposed for BCCs in [7] in order to achieve high throughput continuous decoding; however, the decoding latency required in this case is large. Significantly reduced decoding latency can be obtained with little or no loss in performance by using sliding window decoding, as has been proposed for LDPC convolutional codes [13]–[15]. In this section, we present a novel low-latency sliding window decoding scheme for blockwise SBC codes.

A. Window Decoding

We again use the rate $R = 1/3$ blockwise SBC encoder described in Section II for illustration. The code sequence is $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t$ is given by (1). After transmission over an AWGN channel, the received sequence is $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_t, \ldots)$, where

$$\mathbf{r}_t = \left(r_{t,1}^{(0)}, r_{t,1}^{(1)}, r_{t,1}^{(2)}, r_{t,2}^{(0)}, r_{t,2}^{(1)}, r_{t,2}^{(2)}, \ldots, r_{t,T}^{(0)}, r_{t,T}^{(1)}, r_{t,T}^{(2)}\right).$$

Let $\mathbf{l}^c = (\mathbf{l}_0^c, \mathbf{l}_1^c, \ldots, \mathbf{l}_t^c, \ldots)$ denote the corresponding sequence of received channel log-likelihood ratios (LLRs), where

$$\mathbf{l}_t^c = \left(l_{t,1}^{c,(0)}, l_{t,1}^{c,(1)}, l_{t,1}^{c,(2)}, l_{t,2}^{c,(0)}, l_{t,2}^{c,(1)}, l_{t,2}^{c,(2)}, \ldots, l_{t,T}^{c,(0)}, l_{t,T}^{c,(1)}, l_{t,T}^{c,(2)}\right),$$

which can be decomposed into $\left(\mathbf{l}_t^{c,(0)}, \mathbf{l}_t^{c,(1)}, \mathbf{l}_t^{c,(2)}\right)$, where $\mathbf{l}_t^{c,(j)}$, $j = 0, 1, 2$, represents the T channel LLRs corresponding to the information symbols, the parity output symbols from Encoder 1, and the parity output symbols from Encoder 2, respectively, at time unit t.

[2]This is in contrast to [7], where tail-biting termination was employed at the end of each block.

At the receiver, the channel LLR sequence $\mathbf{l}^c$ is demultiplexed into three streams, $\mathbf{l}^{c,(0)}$, $\mathbf{l}^{c,(1)}$, and $\mathbf{l}^{c,(2)}$, where

$$\mathbf{l}^{c,(j)} = \left(\mathbf{l}_0^{c,(j)}, \mathbf{l}_1^{c,(j)}, \ldots, \mathbf{l}_t^{c,(j)}, \ldots\right), \quad j = 0, 1, 2.$$

The channel LLR inputs to Decoder 1 at time unit t are given by $\mathbf{l}_t^{c,(0)}$, $\mathbf{l}_t^{c,(1)}$, and the permuted parity outputs $\tilde{\mathbf{l}}_{t-1}^{c,(2)}$ from Decoder 2 at time unit $t-1$. Similarly, the channel LLR inputs to Decoder 2 at time unit t are the permuted information inputs $\tilde{\mathbf{l}}_t^{c,(0)}$, the permuted parity outputs $\tilde{\mathbf{l}}_{t-1}^{c,(1)}$ from Decoder 1 at time unit $t-1$, and $\mathbf{l}_t^{c,(2)}$.

Fig. 2 shows a schematic representation of a sliding window decoder. As the transmitted blocks of symbols are received, the first w blocks of channel LLRs are stored in the window, with the initial block on the left, where w is the window size in blocks. Among these blocks, the initial block on the left is referred to as the set of *target symbols*, i.e., the first block of symbols to be decoded. Generally, for a window covering the information blocks from time unit t to time unit $t+w-1$, the T symbols at time unit t are the target symbols, as shown in Fig. 2(a).

There are two types of iterations in the sliding window decoder: *vertical* iterations and *horizontal* iterations. A vertical iteration is a conventional turbo iteration in which extrinsic information on the information bits of a given block is exchanged between component decoders at the same time unit. A horizontal iteration is a forward/backward round trip of information exchanges within the window in which extrinsic information on the parity bits of two successive blocks is exchanged between component decoders at different time units. I_1 and I_2 represent the numbers of vertical (turbo) iterations and horizontal (forward/backward) iterations, respectively. After completing a certain number of vertical iterations and horizontal iterations according to some chosen decoding schedule, the decoding of T target symbols is completed, and then the window shifts one position to the right, as shown in Fig. 2(b).

The decoding procedure within a window is further detailed in Fig. 3, where we use the following notation. For time unit t, $t \in [0, L+\Lambda-1]$, $\mathbf{l}_{\text{Inf}}^c(t)$, $\mathbf{l}_{\text{Inf}}^a(t)$, and $\mathbf{l}_{\text{Inf}}^e(t)$ represent the channel LLRs, the *a priori* LLRs, and the *a posteriori* probability (APP) extrinsic LLRs of the information symbols corresponding to Decoder 1, respectively; $\mathbf{l}_{\text{Pin1}}^c(t)$, $\mathbf{l}_{\text{Pin1}}^a(t)$, and $\mathbf{l}_{\text{Pin1}}^e(t)$ denote the channel LLRs, the *a priori* LLRs, and the extrinsic LLRs of the input parity symbols corresponding to Decoder 1 (the permuted parity outputs from Decoder 2 at time unit $t-1$), respectively; and $\mathbf{l}_{\text{Pout1}}^c(t)$, $\mathbf{l}_{\text{Pout1}}^a(t)$, and $\mathbf{l}_{\text{Pout1}}^e(t)$ denote the channel LLRs, the *a priori* LLRs, and the extrinsic LLRs of the output parity symbols corresponding to Decoder 1, respectively. Similarly, $\tilde{\mathbf{l}}_{\text{Inf}}^c(t)$, $\tilde{\mathbf{l}}_{\text{Inf}}^a(t)$, and $\tilde{\mathbf{l}}_{\text{Inf}}^e(t)$ are the channel LLRs, the *a priori* LLRs, and the extrinsic LLRs of the information symbols corresponding to Decoder 2, respectively; and the Decoder 2 LLRs $\mathbf{l}_{\text{Pin2}}^c(t)$, $\mathbf{l}_{\text{Pin2}}^a(t)$, $\mathbf{l}_{\text{Pin2}}^e(t)$, $\mathbf{l}_{\text{Pout2}}^c(t)$, $\mathbf{l}_{\text{Pout2}}^a(t)$, and $\mathbf{l}_{\text{Pout2}}^e(t)$ are defined analogously to the corresponding LLRs of Decoder 1. Finally, since the window size is w, there are w received blocks, each of length $3T$, in the decoding window at any particular time.

When decoding the set of target symbols at time unit t, the decoding window covers the w blocks at time units $s \in [t, t+w-1]$. We initialize the window decoder as follows. For Decoder 1, we set

$$\begin{aligned}
&\mathbf{l}_{\text{Inf}}^c(s) = \mathbf{l}_s^{c,(0)}, \\
&\mathbf{l}_{\text{Pout1}}^c(s) = \mathbf{l}_s^{c,(1)}, \\
&\mathbf{l}_{\text{Pin1}}^c(s) = \begin{cases} \phi, & s = 0; \\ \mathbf{l}_{s-1}^{c,(2)}\mathbf{P}^{(2)}, & s \in \{t, t+1, \ldots, t+w-1\} \setminus 0, \end{cases}
\end{aligned} \tag{3}$$

and for Decoder 2, we set

$$\begin{aligned}
&\tilde{\mathbf{l}}_{\text{Inf}}^c(s) = \mathbf{l}_s^{c,(0)}\mathbf{P}^{(0)}, \\
&\mathbf{l}_{\text{Pout2}}^c(s) = \mathbf{l}_s^{c,(2)}, \\
&\mathbf{l}_{\text{Pin2}}^c(s) = \begin{cases} \phi, & s = 0; \\ \mathbf{l}_{s-1}^{c,(1)}\mathbf{P}^{(1)}, & s \in \{t, t+1, \ldots, t+w-1\} \setminus 0, \end{cases}
\end{aligned} \tag{4}$$

where ϕ is a large negative constant, since we assume that the second inputs of Encoders 1 and 2 consist of T zeros at initialization (time unit $t = 0$). Also, the *a priori* inputs $\mathbf{l}_{\text{Inf}}^a(0)$, $\mathbf{l}_{\text{Pin1}}^a(0)$, and $\mathbf{l}_{\text{Pout1}}^a(0)$ to Decoder 1 and $\mathbf{l}_{\text{Pin2}}^a(0)$ and $\mathbf{l}_{\text{Pout2}}^a(0)$ to Decoder 2 are set to zero at initialization, while the initial *a priori* input $\tilde{\mathbf{l}}_{\text{Inf}}^a(0)$ to Decoder 2 equals $\mathbf{l}_{\text{Inf}}^e(0)\mathbf{P}^{(0)}$. For all following time units $t = 1, 2, \ldots, L+\Lambda-1$, the channel and *a priori* input LLRs are obtained as indicated in Fig. 3. Throughout decoding, the *a priori* parity input LLRs are initialized with the extrinsic parity output LLRs from the previous time unit as follows:

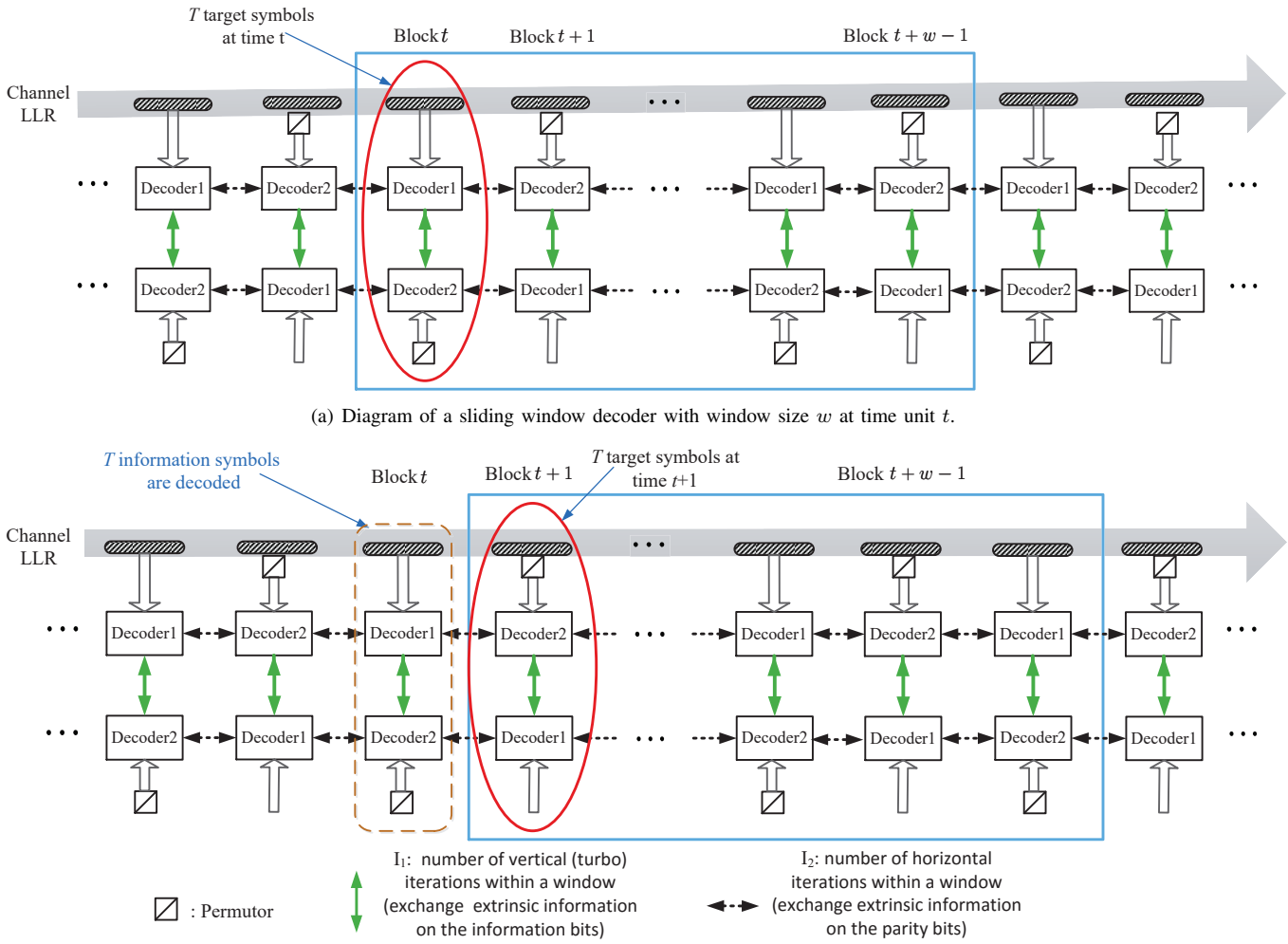
$$\begin{aligned}
\mathbf{l}_{\text{Pin1}}^a(s) &= \mathbf{l}_{\text{Pout2}}^e(s-1)\mathbf{P}^{(2)}, \\
\mathbf{l}_{\text{Pin2}}^a(s) &= \mathbf{l}_{\text{Pout1}}^e(s-1)\mathbf{P}^{(1)},
\end{aligned} \tag{5}$$

for $s = t, t+1, \ldots, t+w-1$, where we assume that, in the case $t = 0$, $\mathbf{l}_{\text{Pout2}}^e(-1) = \mathbf{l}_{\text{Pout1}}^e(-1) = \mathbf{0}$. Also the *a priori* parity output LLRs are initialized with the extrinsic parity input LLRs from the subsequent time unit as follows:

$$\begin{aligned}
\mathbf{l}_{\text{Pout1}}^a(s) &= \begin{cases} \mathbf{l}_{\text{Pin2}}^e(s+1)\left[\mathbf{P}^{(1)}\right]^T, & \text{if block } s+1 \text{ has been previously updated;} \\ \mathbf{0}, & \text{otherwise,} \end{cases} \\
\mathbf{l}_{\text{Pout2}}^a(s) &= \begin{cases} \mathbf{l}_{\text{Pin1}}^e(s+1)\left[\mathbf{P}^{(2)}\right]^T, & \text{if block } s+1 \text{ has been previously updated;} \\ \mathbf{0}, & \text{otherwise,} \end{cases}
\end{aligned} \tag{6}$$

for $s = t, t+1, \ldots, t+w-1$.

The window decoder initializes the first block according to (3)-(6) and starts with vertical decoding of the first block in the window (time unit t in Fig. 3) for I_1 iterations. This vertical decoding phase is identical to decoding a turbo code, except that the APP extrinsic LLRs are calculated for all the code symbols, instead of only for the information symbols. However, during vertical decoding, the extrinsic LLRs $\mathbf{l}_{\text{Inf}}^e(t)$ and $\tilde{\mathbf{l}}_{\text{Inf}}^e(t)$ of the information symbols are active, whereas the extrinsic LLRs $\mathbf{l}_{\text{Pout1}}^e(t)$, $\mathbf{l}_{\text{Pout2}}^e(t)$, $\mathbf{l}_{\text{Pin1}}^e(t)$, and $\mathbf{l}_{\text{Pin2}}^e(t)$ of the parity symbols are inactive. After I_1 vertical iterations, the extrinsic LLRs $\mathbf{l}_{\text{Pout1}}^e(t)$ and $\mathbf{l}_{\text{Pout2}}^e(t)$ of the parity output symbols are then permuted and passed forward to the decoders at time unit $t+1$. After receiving these as *a priori* LLRs, the decoders at time unit $t+1$ are initialized according to (3)-(6), perform I_1 iterations of vertical decoding, and pass the



(a) Diagram of a sliding window decoder with window size w at time unit t.

(b) Diagram of a sliding window decoder with window size w shifting by one time unit.

Fig. 2. Illustration of a sliding window decoder for a rate $R = 1/3$ blockwise SBC code.

extrinsic LLRs $\mathrm{l}^e_{\mathrm{Pout1}}(t+1)$ and $\mathrm{l}^e_{\mathrm{Pout2}}(t+1)$ of their parity output symbols forward to the decoders at time unit $t+2$. This forward process continues until the block at time unit $t+w-1$ finishes vertical decoding.

Following this, the backward exchange process begins and the decoders at time unit $t+w-1$ transmit the permuted extrinsic LLRs $\mathrm{l}^e_{\mathrm{Pin1}}(t+w-1)$ and $\mathrm{l}^e_{\mathrm{Pin2}}(t+w-1)$ of their parity input symbols back to the decoders at time unit $t+w-2$. After receiving these as *a priori* LLRs, the decoders at time unit $t+w-2$ are initialized according to (3)-(6), perform I_1 iterations of vertical decoding, and pass the extrinsic LLRs $\mathrm{l}^e_{\mathrm{Pin1}}(t+w-2)$ and $\mathrm{l}^e_{\mathrm{Pin2}}(t+w-2)$ of their parity input symbols back to the decoders at time unit $t+w-3$. This backward process continues until the block at time t (the first block in the window) finishes vertical decoding. This completes the first horizontal iteration; after this, a new round of horizontal decoding begins and decoding continues in this fashion until I_2 horizontal iterations have been performed, or a stopping rule is met (see Section VI-B).

In the vertical decoding process, the component codes are decoded I_1 times each using the BCJR algorithm [18], where the decoding extends over one block of T information bits. Since the encoders are unterminated after each block, the initial encoder states for the block at time unit t are the same as the final encoder states for the block at time unit $t-1$. The horizontal decoding process (forward and backward) is performed I_2 times. Then hard decisions are made on the target symbols (the first block in the window). After that, the window shifts by one time unit, a new block enters the window, and the decoding process starts again in the new window position. After L information blocks, Λ all-zero termination blocks are used to protect the final information blocks in the sequence, where we typically choose $\Lambda \leq w-1$.

B. Window Decoding Schedules

From the above description, it is clear that the number of vertical iterations I_1 and horizontal iterations I_2 plays an important role in the tradeoff between performance and computational complexity. In this section, we present several window decoding message passing schedules as a means to investigate this tradeoff. A *window decoding schedule* is an algorithm for alternating vertical and horizontal iterations within the window. More specifically, we consider schedules that perform I_2 horizontal iterations within the window and

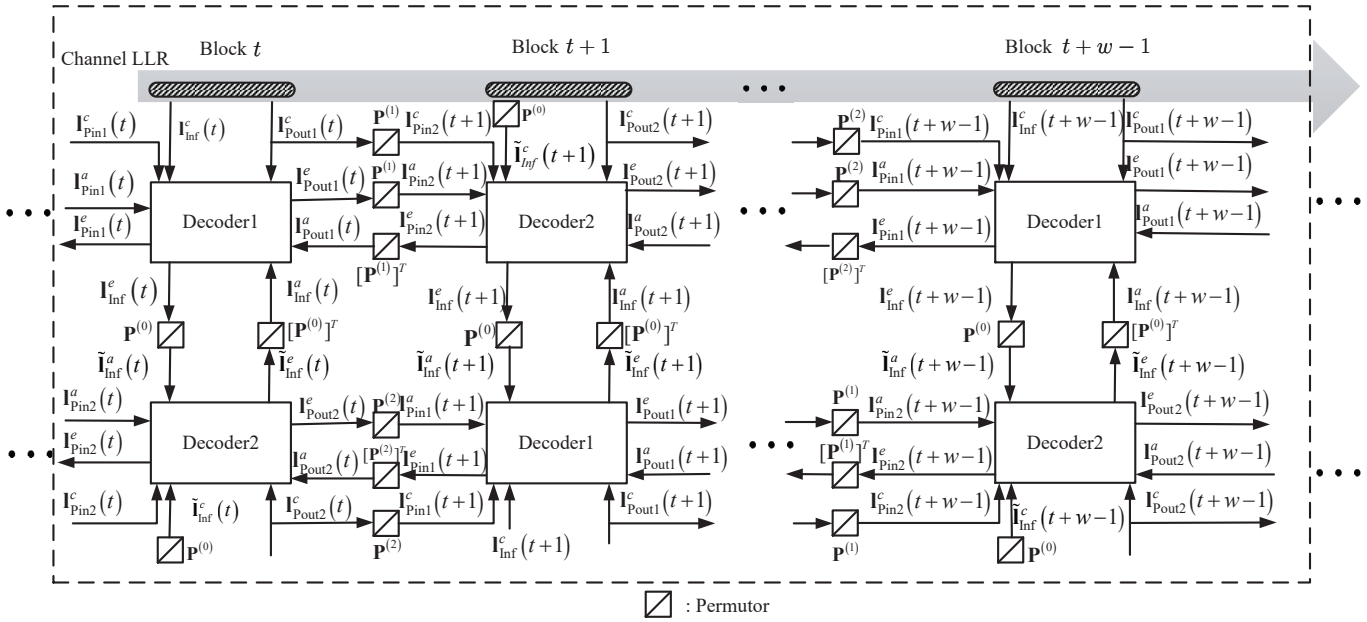


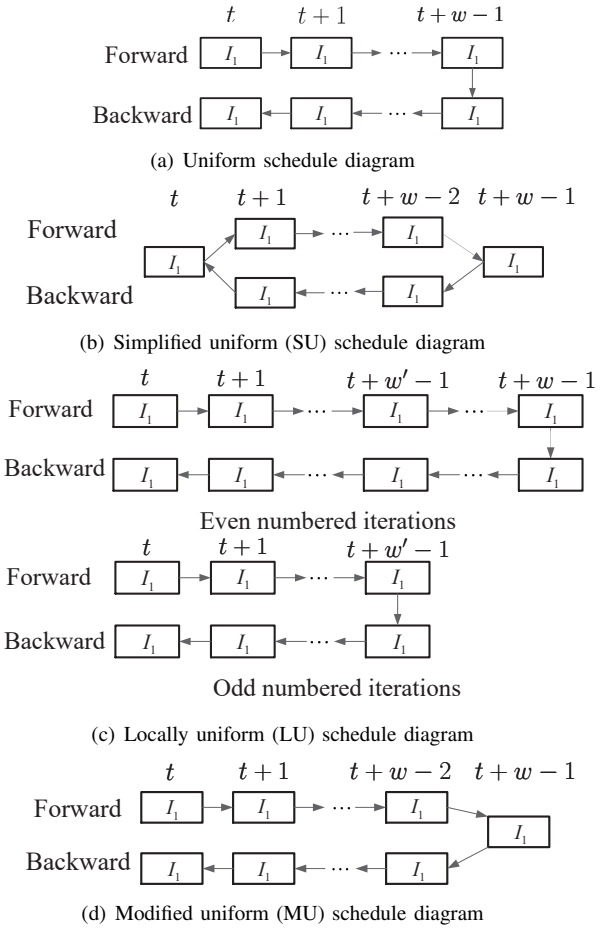
Fig. 3. Decoding within a window for a rate R = 1/3 blockwise SBC code.

I_1 vertical iterations on each block visited during a horizontal iteration. Assume that the window size is w.

1) Uniform Schedule: A diagram of the uniform schedule is shown in Fig. 4(a), where the rectangular boxes represent the blocks at every time unit $s \in [t, t+w-1]$ and I_1 is the number of vertical iterations performed on each block visited. In the forward exchange process, each block in the window successively performs I_1 vertical iterations, beginning with the first block. After I_1 vertical iterations have been performed on the last block in the window, the backward exchange process begins by again performing I_1 vertical iterations on the last block. I_1 vertical iterations are then performed successively on each block from the the last block to the first block. From the diagram we see that, for every horizontal iteration, each block in the window is updated exactly twice. Hence the *total number of iterations* δ in this case is given by $\delta = 2wI_1I_2$.

2) Nonuniform Schedules: In a nonuniform schedule, for every horizontal iteration, the number of times each block is updated varies. Three nonuniform schedules are presented below, but many other nonuniform schedules are clearly possible.

- Simplified uniform (SU) schedule: Compared to the uniform schedule, the end blocks in this schedule only perform I_1 vertical iterations once during each horizontal iteration. A diagram of the SU schedule is shown in Fig. 4(b), where we see that the middle blocks are updated twice as often as the end blocks. (This is the decoding schedule assumed in the description of the operation of the window decoder given at the end of Sec. III-A.) The total number of iterations in this case is given by $\delta = 2\,(w-1)\,I_1I_2$.
- Locally uniform (LU) schedule: In this schedule, during the even numbered horizontal iterations, the decoding schedule is the same as the uniform schedule. During the odd numbered horizontal iterations, however, the forward process stops at block $(w'-1)$, $0 < w' < w$, and the



(a) Uniform schedule diagram

(b) Simplified uniform (SU) schedule diagram

(c) Locally uniform (LU) schedule diagram

(d) Modified uniform (MU) schedule diagram

Fig. 4. Different window decoding schedules.

backward process starts at block $(w'-1)$. A diagram of the LU schedule is shown in Fig. 4(c). The total number of iterations in this case is given by $\delta = I_1 I_2 (w + w')$.

- Modified uniform (MU) schedule: As shown in Fig. 4(d), the difference between the uniform schedule and the MU schedule is that the vertical decoding process operates only once, rather than twice, on the last block in the window during each horizontal iteration. The number of iterations performed during one horizontal iteration is thus I_1 less than for the uniform schedule, i.e., the total number of iterations in this case is given by $\delta = (2w-1) I_1 I_2$.

The idea behind both the LU and MU schedules is to devote a greater fraction of the computational resources to updating the blocks closest to the target symbols [16].

IV. Density Evolution Analysis

We now use density evolution to analyze the asymptotic performance of blockwise SBC codes with window decoding over the binary erasure channel (BEC). The results are then used to guide us in the selection of the window decoding size and a window decoding schedule for large block sizes.

A. Erasure Probabilities of Component Decoders

We again consider the rate $R = 1/3$ blockwise SBC code described in Sec. II, with a rate $R_{cc} = 2/3$ RSC convolutional code as the component code, for illustration. In order to derive an analytical expression for the erasure probability of blockwise SBC codes with window decoding, the extrinsic output erasure probabilities of the component BCJR decoders must be computed.

Denote by $p_{e,out}^{(0)}$, $p_{e,out}^{(1)}$, and $p_{e,out}^{(2)}$ the three extrinsic output erasure probabilities of the component BCJR decoder. In general, these extrinsic output erasure probabilities are functions of the decoder input erasure probabilities p_0, p_1, and p_2, i.e.,

$$\begin{aligned} p_{e,out}^{(0)} &= f_0(p_0, p_1, p_2), \\ p_{e,out}^{(1)} &= f_1(p_0, p_1, p_2), \\ p_{e,out}^{(2)} &= f_2(p_0, p_1, p_2), \end{aligned} \tag{7}$$

where the *transfer functions* $f_0(\cdot)$, $f_1(\cdot)$, and $f_2(\cdot)$ are derived following the method proposed in [9] and [19].

B. Density Evolution for SBC Codes with Window Decoding

With the help of the extrinsic output erasure probabilities of the component decoders, we are able to calculate the evolution of the target symbol erasure probability during the window decoding procedure. Since the component decoder is the same for all iterations, we can use the transfer functions defined above recursively to find the exact decoding probability of erasure for a target symbol after a certain number of iterations.

Without loss of generality, we assume a decoding window from time unit t to time unit $t+w-1$, where w is the window size. During the vertical decoding process, the extrinsic output erasure probabilities after i vertical iterations for the information symbol, the input parity symbol, and the output parity symbol of Decoder 1 at time $s \in [t, t+w-1]$ can be obtained as

$$p_{D_1,\text{Inf}}^{(i,s)} = f_{D_1,\text{Inf}}\left(q_{D_2,\text{Inf}}^{(i-1,s)}, q_{D_2,\text{Pout}}^{(I_1,s-1)}, q_{D_2,\text{Pin}}^{(I_1,s+1)}\right) \tag{8a}$$

$$p_{D_1,\text{Pin}}^{(i,s)} = f_{D_1,\text{Pin}}\left(q_{D_2,\text{Inf}}^{(i-1,s)}, q_{D_2,\text{Pout}}^{(I_1,s-1)}, q_{D_2,\text{Pin}}^{(I_1,s+1)}\right) \tag{8b}$$

$$p_{D_1,\text{Pout}}^{(i,s)} = f_{D_1,\text{Pout}}\left(q_{D_2,\text{Inf}}^{(i-1,s)}, q_{D_2,\text{Pout}}^{(I_1,s-1)}, q_{D_2,\text{Pin}}^{(I_1,s+1)}\right), \tag{8c}$$

where

$$q_{D_2,\text{Inf}}^{(i-1,s)} = \epsilon \cdot p_{D_2,\text{Inf}}^{(i-1,s)} \tag{9a}$$

$$q_{D_2,\text{Pout}}^{(I_1,s-1)} = \epsilon \cdot p_{D_2,\text{Pout}}^{(I_1,s-1)} \tag{9b}$$

$$q_{D_2,\text{Pin}}^{(I_1,s+1)} = \epsilon \cdot p_{D_2,\text{Pin}}^{(I_1,s+1)}. \tag{9c}$$

Here $f_{D_1,Inf}$, $f_{D_1,Pin}$, and $f_{D_1,Pout}$ are the extrinsic output erasure probability transfer functions from (7) for the information symbol, the input parity symbol, and the output parity symbol of component Decoder 1 (D_1), respectively; ϵ denotes the erasure probability of the channel; and I_1 is the maximum number of vertical iterations. Because of the symmetric design, the extrinsic output erasure probability update equations for component Decoder 2 (D_2) are identical to those of decoder D_1 after interchanging D_1 and D_2 in (8)-(9).

During the horizontal decoding process, in the forward exchange, D_1 (resp. D_2) at time unit s transmits the extrinsic erasure probability of the output parity symbol to D_2 (D_1) at time unit $s+1$ as the *a priori* erasure probability of the input parity symbol. In the backward exchange, D_1 (D_2) at time unit s transmits the extrinsic erasure probability of the input parity symbol to D_2 (D_1) at time unit $s-1$ as the *a priori* erasure probability of the output parity symbol. As mentioned in Section III, the second inputs of Encoder 1 and 2 are all zeros at time unit 0. Therefore, in the case $t = 0$, $q_{D_1,\text{Pin}}^{(I_1,-1)} = q_{D_2,\text{Pin}}^{(I_1,-1)} = 0$ and $q_{D_1,\text{Pout}}^{(I_1,-1)} = q_{D_2,\text{Pout}}^{(I_1,-1)} = \epsilon$.

Finally, the decoding erasure probability of a target symbol in the current decoding window $[t, t+w-1]$ for blockwise SBC codes after j iterations of the horizontal decoding process is given by

$$p_{e,t} = \epsilon \cdot p_{D_1,\text{Inf}}^{(j,t)} \cdot p_{D_2,\text{Inf}}^{(j,t)}. \tag{10}$$

C. Results and Discussion

Using the density evolution procedure described above, we first measure the impact of the window size on decoding performance. Then we evaluate different window decoding schedules in terms of decoding complexity. In particular, given the channel erasure probability and a target erasure probability, the schedule which uses the fewest number of iterations is taken to be optimal. As an example, we consider a $R = 1/3$ blockwise SBC code with two identical rate $R_{cc} = 2/3$, 4-state RSC component encoders whose generator matrix is given by

$$G_1(D) = \begin{pmatrix} 1 & 0 & \frac{1}{1+D+D^2} \\ 0 & 1 & \frac{1+D^2}{1+D+D^2} \end{pmatrix}. \tag{11}$$

The BEC thresholds ϵ_w for this blockwise SBC code using window decoding with different window sizes and the modified uniform schedule are shown in Table I. We observe that there is no significant improvement in the threshold with increasing window size beyond $w = 3$. This suggests that,

TABLE I
THE BEC THRESHOLD ϵ_w OF A RATE $R = 1/3$ BLOCKWISE SBC CODE USING WINDOW DECODING WITH DIFFERENT WINDOW SIZES w.

w	2	3	4	5	6	7
ϵ_w	0.652703	0.655166	0.655367	0.655384	0.655386	0.655386

TABLE II
REQUIRED NUMBER OF ITERATIONS δ FOR A RATE $R = 1/3$ BLOCKWISE SBC CODE.

Schedule	I_1	I_2	δ	Schedule	I_1	I_2	δ
Uniform	1	11	66	LU ($w' = 2$)	1	11	55
Uniform	2	7	84	LU ($w' = 2$)	2	7	70
Uniform	3	6	108	LU ($w' = 2$)	3	6	90
SU	1	18	72	MU	1	11	55
SU	2	10	80	MU	2	7	70
SU	3	7	84	MU	3	6	90

for large block sizes, blockwise SBC codes using window decoding with $w = 3$ will achieve good performance, which is consistent with the simulation results presented in the following section.

Table II shows the total required number of iterations δ with different window decoding schedules for channel erasure probability $\epsilon = 0.65$, window size $w = 3$, and target symbol erasure probability $\hat{\varepsilon} = 10^{-9}$. The minimum possible vertical iteration number I_1 is 1 for any window decoding schedule, and we see that this is sufficient to achieve good performance in all cases. Consider the uniform schedule as an example. Here, the number of iterations per horizontal iteration is $6I_1$. For $I_1 = 1$, the number of horizontal iterations needed to achieve $\hat{\varepsilon} = 10^{-9}$ is $I_2 = 11$, and hence the total required number of iterations is $\delta = 6I_1 \cdot I_2 = 66$. For $I_1 = 2$, the required number of horizontal iterations is now $I_2 = 7$, and thus the total number of iterations in this case increases to $\delta = 6I_1 \cdot I_2 = 84$. For $I_1 = 3$, the total number of iterations grows further to $\delta = 6I_1 \cdot I_2 = 108$. Consequently, for large block sizes, we should pick $I_1 = 1$ to obtain the best performance/complexity tradeoff.

In addition, we see that, for fixed I_1 with window decoding, the MU schedule and the LU schedule use the same number of iterations. Furthermore, they use the fewest number of iterations among all the schedules to achieve $\hat{\varepsilon} = 10^{-9}$. For example, for $I_1 = 1$, a total of $\delta = 55$ iterations is needed with the MU or LU schedule (note that the total number of vertical iterations per horizontal iteration is $5I_1$ for both these schedules). However, $\delta = 66$ iterations are needed with the uniform schedule and $\delta = 72$ iterations are needed with the SU schedule. Therefore, in this case, the MU or LU schedule is the best choice for large block sizes. For larger w, we expect the LU schedule to be the most efficient due to its flexibility.

V. PERFORMANCE OF BLOCKWISE SBC CODES WITH SLIDING WINDOW DECODING

In this section, some examples are given to illustrate the performance of blockwise SBC codes with sliding window decoding over the AWGN channel with binary phase-shift keying (BPSK) signaling.

We consider the rate $R = 1/3$ blockwise SBC code with two identical 4-state RSC component encoders whose generator matrix is given by (11), 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. We assume that a transmission consists of an information sequence of length $50T$, i.e. 50 information blocks, plus $\Lambda = 1$ all-zero termination block of length T, so that the overall frame length is $152T$ and the actual rate is $\tilde{R} = 0.329$ (see (2)).

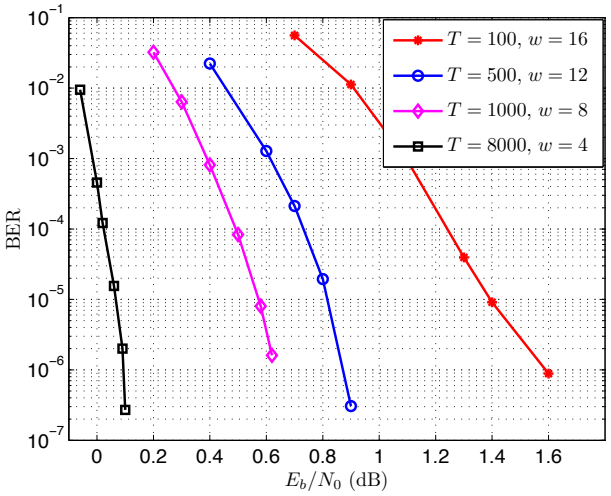


Fig. 5. BER performance of rate $R = 1/3$ blockwise SBC codes with window decoding and different permutor sizes T. The vertical and horizontal iteration numbers are $I_1 = 5$ and $I_2 = 30$, respectively.

The bit error rate (BER) performance of rate $R = 1/3$ blockwise SBC codes with the SU sliding window decoding schedule is shown in Fig. 5, where the permutor sizes are $T = 100$, 500, 1000, and 8000, and the corresponding window sizes are $w = 16$, 12, 8, and 4, respectively. The results indicate how the performance scales with permutor size, assuming we choose a sufficiently large window. We found in general that a smaller T requires a larger w. This is because the base turbo code with a small block size is weak, so larger windows are needed to protect the target symbols in this case. We also see that the performance with window decoding improves as we increase the size of the block permutors, as expected. For code rate $R = 1/3$, the gap to capacity is about 0.5 dB with permutor size $T = 8000$ and $w = 4$, and no error floor is observed.

In addition to BER performance, the decoding latency introduced by channel coding is a crucial factor in the design of a practical communication system. For the rate $R = 1/3$ blockwise SBC code, the decoding latency of the sliding window decoder is given by

$$\tau = 3Tw \tag{12}$$

symbols, where T and w are design parameters that can be chosen to satisfy a latency constraint.

The bit signal-to-noise ratio E_b/N_0 required to achieve a BER of 10^{-5} as a function of decoding latency is shown in Fig. 6. We observe that the performance of blockwise SBC codes with a fixed permutor size improves as the window size w increases; however, it does not improve much beyond a

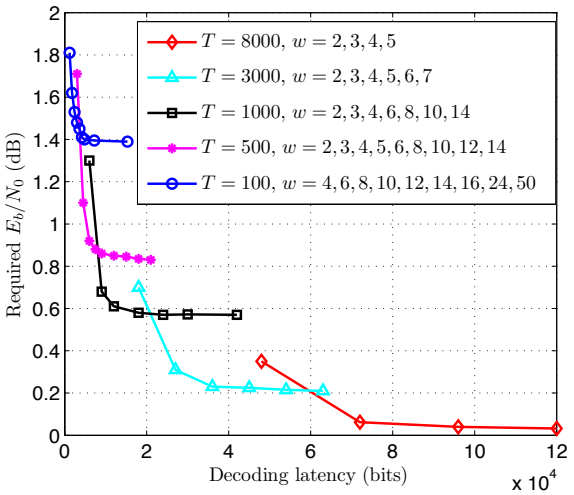


Fig. 6. Required E_b/N_0 to achieve a BER of 10^{-5} for the rate $R = 1/3$ blockwise SBC code as a function of decoding latency.

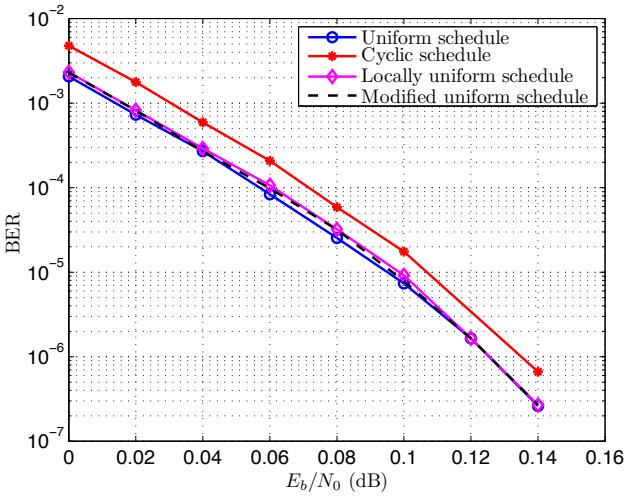


Fig. 7. BER performance of rate $R = 1/3$ blockwise SBC codes with window decoding using different decoding schedules for the same number of vertical and horizontal iterations. The window size $w = 3$, the block permutor size $T = 8000$, and the vertical and horizontal iteration numbers are $I_1 = 1$ and $I_2 = 20$, respectively. For the locally uniform schedule, $w' = 2$.

certain window size. Moreover, beyond a certain latency, using a larger permutor size T with a smaller window size w gives better performance.

For a fixed number of iterations, the performance of blockwise SBC codes with window decoding and different decoding schedules is compared in Fig. 7. The uniform, SU, LU, and MU schedules are examined with $I_1 = 1$ and $I_2 = 20$, block size $T = 8000$, and window size $w = 3$, where w was chosen according to the density evolution results from Table I for large block sizes. In this case, the total number of iterations for window size $w = 3$ is 120, 80, 100, and 100, respectively. We see that the performance of the window decoder with the locally uniform or modified uniform schedule is almost identical to that of the uniform schedule, but this performance is achieved with fewer iterations. This result is consistent with the density evolution results from Table II, where the locally uniform and modified uniform schedules were found to be the most computationally efficient for large block sizes. We also see that the SU schedule has the worst performance, since the target symbols are updated less often during a round of horizontal decoding than with the other schedules. Finally, we note that the performance of the blockwise SBC code with the locally uniform or modified uniform schedule shown in Fig. 7 is about 0.2 dB better than the results published in [7], which used pipeline decoding with the same total number of turbo iterations, i.e., 100, but with a much higher decoding latency.[3]

VI. Computational Complexity

In this section, we investigate the computational complexity of SBC codes with sliding window decoding and propose two stopping rules, one based on cross entropy and the other on LLR magnitudes, to reduce computational complexity.

[3]The reason for the better performance of the sliding window decoder when the number of iterations is fixed is that it uses its iterations more efficiently, since it confines them to a finite-size window, whereas the pipeline decoder effectively implements the flooding schedule over an entire frame.

A. Computational Complexity Analysis

The computational complexity of a sliding window decoder can be analyzed by enumerating the number of operations required to decode a block of T information bits. As illustrated above, window decoding of blockwise SBC codes requires a set of w turbo decoders operating within a sliding window. The factors determining the computational complexity of a turbo decoder include the trellis complexity, the number of iterations allowed, and the number of constituent decoders. We now enumerate the number of multiplications (divisions), additions (subtractions), and comparisons in the BCJR algorithm for a single trellis section of target bits (the first block of information bits in the window).

Assume that we take a rate $R_{cc} = k/n$ RSC code (k inputs, n outputs) with S states and B branches leaving each state in its trellis representation as the component code. For the Log-Map BCJR algorithm with forward recursion values α, backward recursion values β, and branch metrics γ (see [20] for details), the following operations are required:

- For the forward recursion, $S \cdot (B-1)$ comparisons and $S \cdot (2B-1)$ additions are required. To normalize the α's, an additional $S-1$ comparisons and S additions are needed. Hence, the total number of required computations for the forward recursion is $2S \cdot B$ additions and $S \cdot B - 1$ comparisons per trellis section;
- The required number of computations for the β's is the same as for the α's, i.e., $2S \cdot B$ additions and $S \cdot B - 1$ comparisons per trellis section;
- For the branch metric γ, $S \cdot B \cdot (n+1)$ multiplications and $S \cdot B \cdot n$ additions are required;
- For the computation of the APP values, $S \cdot B - 2$ comparisons and $2S \cdot B$ additions are required;
- For the computation of the extrinsic LLRs, 1 addition is required;
- For one trellis section, one vertical iteration, and per bit,

TABLE III
COMPUTATIONS C_{TS} PER BIT FOR ONE TRELLIS SECTION AND ONE ITERATION.

	Multiplications	Additions	Comparisons
α	0	$2S \cdot B$	$S \cdot B - 1$
β	0	$2S \cdot B$	$S \cdot B - 1$
γ	$S \cdot B \cdot (n+1)$	$S \cdot B \cdot n$	0
APP values	0	$2S \cdot B$	$S \cdot B - 2$
Extrinsic values	0	1	0

the number of computations (multiplications, additions, and comparisons) required for a Log-MAP decoder, C_{TS}, is summarized in Table III.

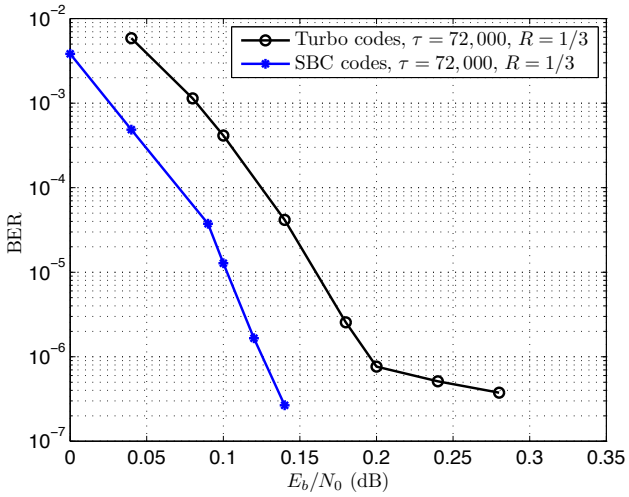
Therefore, for a sliding window decoder with Tw information bits in a window, number of vertical iterations I_1, number of horizontal iterations I_2, component code rate $R_{cc} = k/n$, and code rate R, the computational complexity of the overall decoder can be written as

$$C_{total} = 2 \cdot \delta \cdot \mathrm{T} \cdot n \cdot C_{TS} \tag{13}$$

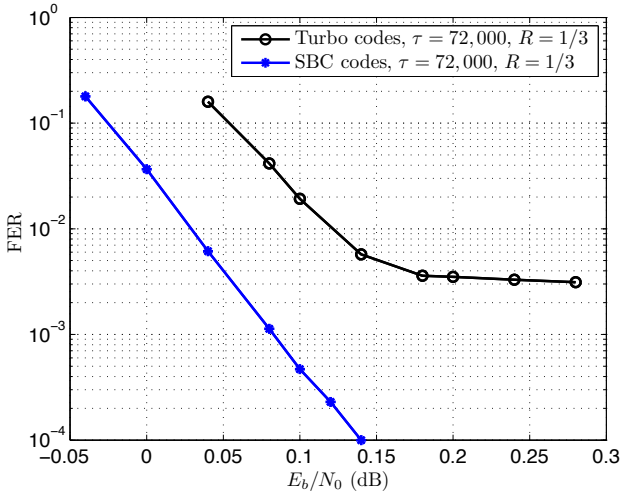
where δ is the total number of iterations per bit for a given window decoding schedule as defined in Sec. III-B. Note that, as mentioned previously, the component decoders calculate the LLRs of the information bits and the parity bits, and hence a total of $T \cdot n$ LLRs must be calculated in each block.

We now consider a performance vs. complexity comparison of a blockwise SBC code and a turbo code with the same rate and decoding latency τ. For the turbo code, the computational complexity is given by $2 \cdot \delta' \cdot T' \cdot n' \cdot C'_{TS}$, where δ' is the number of turbo iterations, T' is the number of information bits in a block, n' is the total number of output bits of each trellis section, and C'_{TS} is the number of required computations per bit for one trellis section and one iteration. Both the BER and the frame error rate (FER) performance of a rate $R = 1/3$ blockwise SBC code with 4-state, $R_{cc} = 2/3$ component codes and the rate $R = 1/3$ turbo code used in the CDMA2000 industry standard [21] with a randomly chosen permutor and 8-state $R_{cc} = 1/2$ component codes is shown in Fig. 8, where we choose $T' = Tw$ so that the decoding latency $\tau = T' = Tw$ is the same in both cases. We see that, in the waterfall region, the SBC code outperforms the turbo code by 0.075 dB at a BER of 10^{-6} and by 0.095 dB at an FER of 10^{-2}. Moreover, there is no visible error floor for the SBC code[4], while an error floor appears for the turbo code beginning at a BER of 10^{-6} and an FER of 5×10^{-3}. In terms of performance, SBC codes are particularly attractive for moderate and high latency applications; however, their advantage over turbo codes disappears for smaller latencies since, referencing the results of Fig. 5, we see that relatively large windows are required for small permutor sizes. With respect to computational complexity, since $S \cdot B$ is the same in both cases, that is $C'_{TS} = C_{TS}$, but $n = 3$ for the SBC code and $n' = 2$ for the turbo code, $n \cdot C_{TS}$ for the SBC code is larger than $n' \cdot C'_{TS}$ for the turbo code. Also $\delta = 2wI_1I_2 = 72$ for the SBC code, whereas the number of iterations $\delta' = 20$ is smaller for the turbo code.

[4]This is expected because of the linear distance growth property of these codes.



(a) BER performance



(b) FER performance

Fig. 8. BER and FER performance of a blockwise SBC code and a turbo code with the same rate R and decoding latency τ. For the SBC code, a uniform decoding schedule is used where the parameters chosen are $w = 3$, $T = 8000$, $I_1 = 1$, and $I_2 = 12$. For the turbo code, the information block length is $T' = 24,000$ and the number of iterations is $\delta' = 20$.

In summary, we get improved waterfall performance and no error floors with SBC codes when compared to turbo codes, at the expense of some additional computation. It is important to note, though, that the performance vs. complexity tradeoff will change if the number of iterations is reduced or if decoder stopping rules are employed, but that additional iterations will not change the performance. We will see in the following that the complexity C_{total} of SBC codes can be reduced without significant performance degradation by considering efficient stopping rules.

B. Stopping Rules

Since the computational complexity of a sliding window decoder depends on the number of iterations performed in a window, stopping rules can be used to detect decoder convergence and thereby reduce the computational complexity. Here we propose two stopping rules: one based on the cross

entropy (CE) [17] of the distribution of APP values at the outputs of two decoders; the other based on the magnitudes of the LLRs of the target information bits.

1) Cross entropy based stopping rule: The CE stopping rule is based on the difference between the APP values of the target information bits (the first block in the window) at the output of the two decoders after successive horizontal iterations. Let $Q_{(i)}(u_l)$ and $P_{(i)}(u_l)$ represent the APP distributions at the output of Decoders 1 and 2 at the i^{th} horizontal iteration, respectively, and let $L^{(Q)}_{(i)}(u_l)$ and $L^{(P)}_{(i)}(u_l)$ represent the corresponding APP values. We write the difference in the two soft outputs as (see [17] for details)

$$\Delta L^{(P)}_{e(i)}(u_l) \triangleq L^{(P)}_{(i)}(u_l) - L^{(Q)}_{(i)}(u_l) = L^{(P)}_{e(i)}(u_l) - L^{(P)}_{e(i-1)}(u_l); \tag{14}$$

that is, $\Delta L^{(P)}_{e(i)}(u_l)$ represents the difference in the APP extrinsic LLRs $L^{(P)}_{e(i)}(u_l)$ of Decoder 2 in two successive horizontal iterations. We next define (see [17] for details)

$$T(i) \triangleq \sum_{l=1}^{T} \frac{\left|\Delta L^{(P)}_{e(i)}(u_l)\right|^2}{e^{\left|L^{(Q)}_{(i)}(u_l)\right|}} \tag{15}$$

as the approximate value of the CE at horizontal iteration i. $T(i)$ can be computed after each horizontal iteration.

Experience has shown that once convergence is achieved, $T(i)$ drops by a factor of at least 10^{-3} compared with its initial value $T(0)$. Hence if $T(i)$ is less than $\eta T(0)$ after i horizontal iterations, where $\eta \in [10^{-3}, 10^{-6}]$ is a user selected parameter, decoding stops and the window shifts.

2) LLR magnitude based stopping rule: The LLR magnitude stopping rule is based on the convergence of the cumulative LLR magnitudes of the target information bits. During the decoding process, as decoding converges to the correct codeword, the LLRs of the T target symbols typically tend to large positive or negative values and become more stable as the number of iterations increases. We define the total LLR magnitude of the T target symbols at horizontal iteration i, denoted $\lambda^{(i)}_{tot}$, as

$$\lambda^{(i)}_{tot} = \sum_{l=1}^{T} \left|\lambda^{(i)}(l)\right|, \tag{16}$$

where $\left|\lambda^{(i)}(l)\right|$ is the LLR magnitude of target information bit l at horizontal iteration i. This value typically increases iteration by iteration and eventually converges to a stable value; consequently, the difference of the $\lambda^{(i)}_{tot}$ values in two successive horizontal iterations typically decreases. We write the absolute value of the difference of the $\lambda^{(i)}_{tot}$ values in two successive horizontal iterations as

$$\Delta\lambda^{(i)}_{tot} = \left|\lambda^{(i)}_{tot} - \lambda^{(i-1)}_{tot}\right|. \tag{17}$$

The LLR magnitude based stopping rule uses two user selected parameters: one is a magnitude threshold, denoted by θ, used to determine whether the difference of two successive LLR magnitudes is sufficiently small; the other is a depth factor, denoted by M, whose role is to ensure that the diminishing LLR magnitude difference is consistent and actually reflects decoding convergence. The two parameters together determine whether further iterations should be performed. Hence if $\lambda^{(i)}_{tot}$ is less than θ for M successive horizontal iterations, decoding stops and the window shifts.

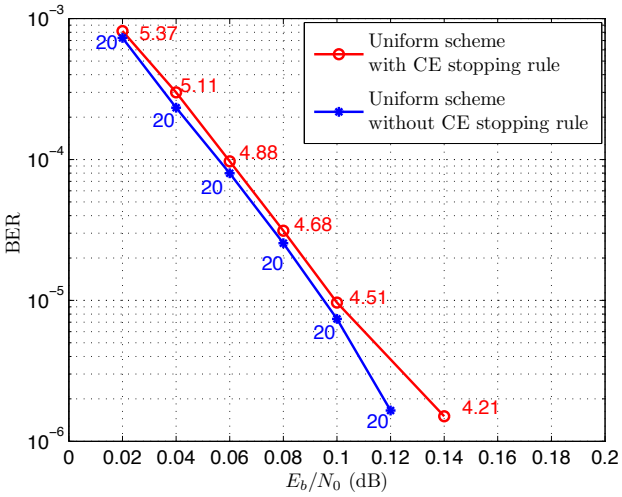


Fig. 9. Performance comparison of rate $R = 1/3$ blockwise SBC codes using a uniform window decoding schedule with and without the CE based stopping rule. The numbers next to the simulation points are the average number of horizontal iterations. The parameters chosen are $w = 3$, $T = 8000$, $I_1 = 1$, $I_2 = 20$, and $\eta = 10^{-6}$.

3) Numerical Examples: To examine the efficiency of the proposed stopping rules, we performed simulations of blockwise SBC codes with code rate $1/3$ and a uniform decoding schedule with $I_1 = 1$. The performance comparison of the blockwise SBC codes with and without the CE stopping rule is shown in Fig. 9. With the CE stopping rule ($\eta = 10^{-6}$), the performance is less than 0.01 dB worse than without the stopping rule at a $BER = 10^{-5}$, but the average number of horizontal iterations is greatly reduced. For $\frac{E_b}{N_0} = 0.1$ dB, where the $BER = 10^{-5}$, the average number of horizontal iterations $\bar{I}_2 = 4.5$, and hence the computational complexity with the CE stopping rule is $C_{total} = 2\delta \cdot T \cdot n \cdot C_{TS} = 2{\cdot}2{\cdot}I_1{\cdot}\bar{I}_2{\cdot}w{\cdot}T{\cdot}n{\cdot}C_{TS} = 162T{\cdot}C_{TS}$, while the computational complexity without the stopping rule is $C_{total} = 720T \cdot C_{TS}$.

The BER performance of SBC codes with the LLR magnitude based stopping rule is shown (for $\theta = 80$ and $M = 2$) in Fig. 10. Similar to the CE stopping rule, we see that the average number of horizontal iterations is greatly reduced with negligible performance loss. For $\frac{E_b}{N_0} = 0.1$ dB, where the $BER = 10^{-5}$, the average number of horizontal iterations $\bar{I}_2 = 8$, and hence the computational complexity with the LLR magnitude stopping rule and the uniform decoding schedule is $C_{total} = 288T \cdot C_{TS}$.

Based on these numerical examples, there appears to be little difference between the two stopping rules, with the CE stopping rule needing somewhat fewer iterations but the LLR magnitude stopping rule giving slightly better performance.

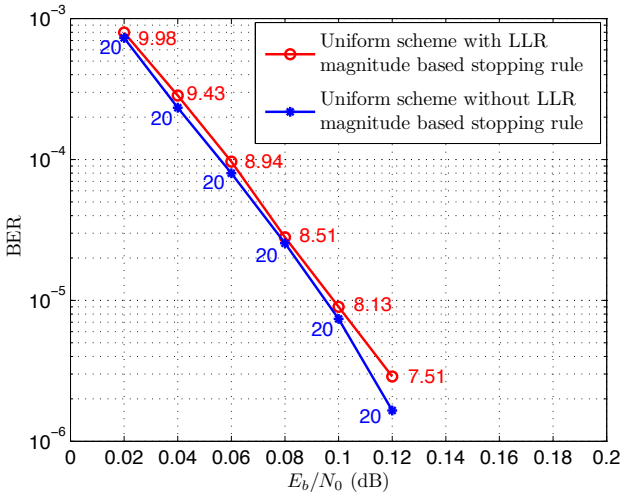


Fig. 10. Performance comparison of rate $R = 1/3$ blockwise SBC codes using a uniform window decoding schedule with and without the LLR magnitude based stopping rule. The numbers next to the simulation points are the average number of horizontal iterations. The parameters chosen are $w = 3$, $T = 8000$, $I_1 = 1$, $I_2 = 20$, $M = 2$, and $\theta = 80$.

VII. Rate-compatible blockwise SBC codes

In this section, we discuss the construction of rate-compatible blockwise SBC codes [5], obtained by puncturing a low rate SBC mother code to achieve higher code rates. We first introduce the puncturing technique and present simulation results illustrating the performance of punctured SBC codes. We then introduce a set of rate-compatible SBC codes obtained from a lower rate SBC mother code.

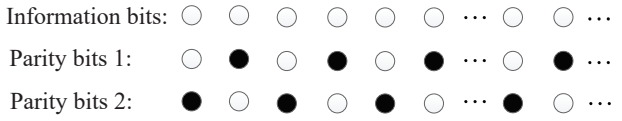
A. Puncturing Technique

As an example, we take the $R = 1/3$ blockwise SBC code with rate $R_{cc} = 2/3$ component codes as the mother code. Periodic puncturing is then used to obtain higher code rates. The puncturing patterns used to obtain $R = 1/2$ and $R = 2/3$ blockwise SBC codes are shown in Fig. 11. Due to the fact that the blockwise SBC mother code is systematic, we puncture only parity bits. At each time instant, one information bit and two parity bits come out of the SBC encoder. In order to obtain an SBC code with rate $R = 1/2$, one parity bit at each time instant is punctured in an alternating pattern (see Fig. 11(a)). In order to obtain an SBC code with rate $R = 2/3$, six parity bits in every four time instants are punctured (see Fig. 11(b)). Additional puncturing to produce a code rate R greater than the rate R_{cc} of the component codes does not produce good results.

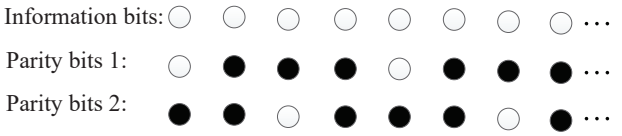
B. Numerical Examples

We now investigate the performance of punctured rate-compatible blockwise SBC codes with window decoding. Using the rate $R = 1/3$ SBC code with $T = 8000$ as the mother code, we obtain rate-compatible SBC codes with code

[5]The puncturing patterns are not nested, and hence the codes are not rate-compatible in the strictest sense.



(a) Puncturing pattern used to obtain a blockwise SBC code with rate $1/2$.



(b) Puncturing pattern used to obtain a blockwise SBC code with rate $2/3$.

Fig. 11. Puncturing patterns based on the mother code with rate $1/3$. White circles indicate transmitted bits and black circles correspond to punctured (non-transmitted) bits.

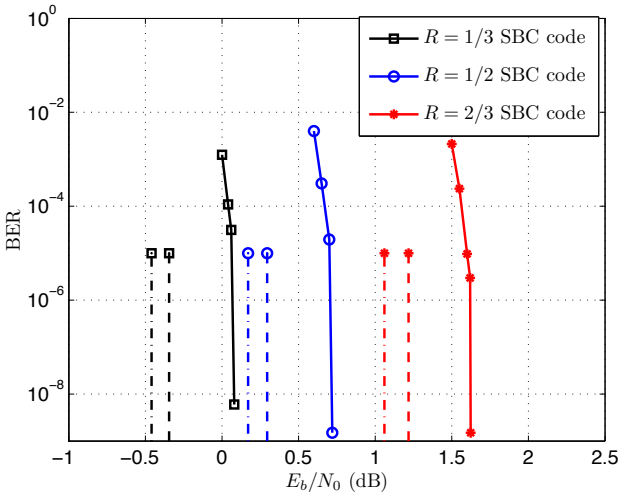


Fig. 12. Performance comparison of rate-compatible SBC codes with window decoding obtained by periodic puncturing. Shown for comparison are capacity limits (dot-dash lines) and finite-length limits (dotted lines) [22], [23].

rates of 1/2 and 2/3 by periodic puncturing as illustrated in Fig. 11. Their BER performance using a uniform window decoding schedule with $I_1 = 1$, $I_2 = 20$, and $w = 3$ is shown in Fig. 12. The decoding latencies of the rate-compatible SBC codes with rates $1/3$, $1/2$, and $2/3$ are $72,000$, $48,000$, and $36,000$, respectively. At a BER of 10^{-5}, the rate-compatible SBC codes with code rates of 1/3, 1/2, and 2/3 perform about 0.56 dB, 0.58 dB, and 0.62 dB away from the Shannon limit, respectively, and they show no visible sign of an error-floor down to a BER of 10^{-8}. It can also be seen that, at a BER of 10^{-8}, the rate-compatible SBC codes are less than 0.5 dB away from the finite-length limit, which is calculated according to [22], [23] for a code length equal to the decoding latency of the SBC codes.

Next, the performance of a rate $R = 0.495$ blockwise SBC code is compared to a rate $R = 0.499$ turbo code and a rate $R = 0.49$ LDPC convolutional (spatially coupled) code, all with the same decoding latency τ, in Fig. 13. The blockwise SBC code is obtained by puncturing the $R = 1/3$ SBC mother code with 4-state, $R_{cc} = 2/3$ component codes, the turbo code is obtained by puncturing the rate $R = 1/3$ turbo code with 8-state, R_{cc} = 1/2 component codes used in the CDMA2000 standard, and the LDPC convolutional code is

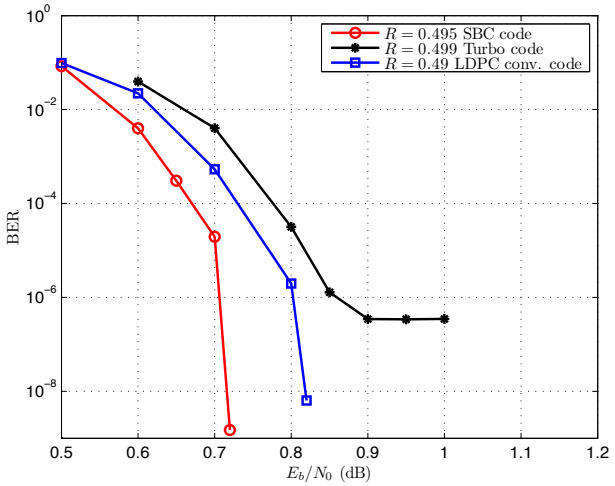


Fig. 13. BER performance of a blockwise SBC code, a turbo code, and an LDPC convolutional code, all with the same decoding latency $\tau = 48,000$ symbols.

randomly constructed based on the $(4, 8)$-regular protograph with base matrix $\mathbf{B} = [4, 4]$, where the component base matrices are $\mathbf{B}_0 = \mathbf{B}_1 = \mathbf{B}_2 = \mathbf{B}_3 = [1, 1]$ (see [24] for details of the protograph construction). For the SBC code, a uniform window decoding schedule was used, where the window size $w = 3$ and the block permutor size $T = 8000$. For the turbo code, the information block length is 24,000, and for the LDPC convolutional code, the window size, the protograph lifting factor, and the coupling length are 12, 2000, and 100, respectively (see [24] for details). We observe that, for the same decoding latency $\tau = 48,000$ bits, the blockwise SBC code outperforms the LDPC convolutional code by about 0.1 dB at a BER of 10^{-6}. Moreover, the blockwise SBC code outperforms the turbo code used in the CDMA2000 standard by about 0.15 dB and displays no visible error floor, which is expected because of the linear distance growth property of SBC codes.

C. Rate-compatible SBC codes obtained from a higher rate mother code

In previous sections, we discussed a rate $R = 1/3$ blockwise SBC code with two identical rate $R_{cc} = 2/3$, 4-state RSC component convolutional codes. In this section, we consider a rate $R = 1/2$ blockwise SBC code with two identical rate $R_{cc} = 3/4$, 8-state RSC component convolutional codes. The generator matrix of the component encoders is given by

$$G_2(D) = \begin{pmatrix} 1 & 0 & 0 & \frac{1+D+D^3}{1+D^3} \\ 0 & 1 & 0 & \frac{1+D^2+D^3}{1+D^3} \\ 0 & 0 & 1 & \frac{D+D^3}{1+D^3} \end{pmatrix},$$

and the encoder is shown in Fig. 14 (compare to the encoder in Fig. 1). The difference is that the information block $\mathbf{u}_t$ of length $2T$ is divided into two sub blocks $\mathbf{u}_{t,1}$ and $\mathbf{u}_{t,2}$, each of length T, which are two of the inputs to each component encoder. In this case, the actual rate of the resulting SBC codes, including the termination, is $\tilde{R} = \frac{1}{2}\frac{L}{L+\Lambda/2} \underset{L\to\infty}{\to} \frac{1}{2}$, where L and Λ are defined in Section II.

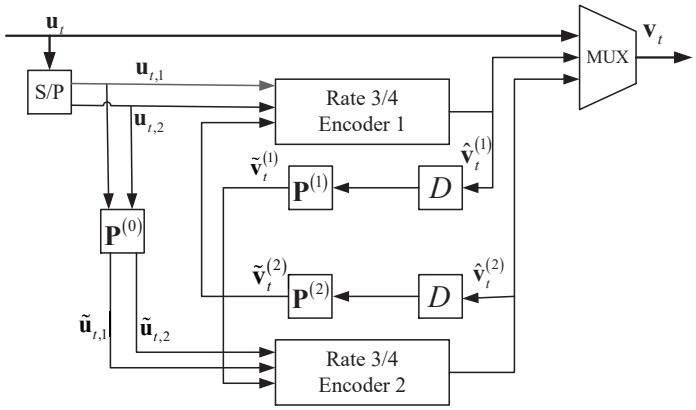
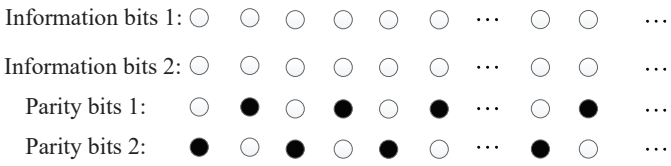
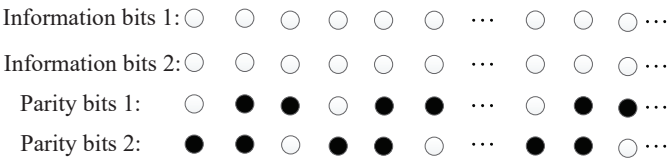


Fig. 14. Encoder for a rate $R = 1/2$ blockwise SBC code.



(a) Puncturing pattern to obtain a blockwise SBC code with rate $2/3$.



(b) Puncturing pattern to obtian a blockwise SBC code with rate $3/4$.

Fig. 15. Puncturing patterns based on the mother code with rate $1/2$.

The puncturing patterns used to obtain $R = 2/3$ and $R = 3/4$ blockwise SBC codes are shown in Figs. 15(a) and 15(b), respectively. By puncturing, we obtain a rate-compatible set of blockwise SBC codes with code rates of $1/2$, $2/3$, and $3/4$. In Fig. 16, we show the BER performance of this rate-compatible set with $T = 9000$. A uniform window decoding schedule was used with $I_1 = 1$, $I_2 = 20$, and $w = 3$. It is observed that, at a BER of 10^{-5}, the rate-compatible SBC codes with rates $1/2$, $2/3$, and $3/4$ perform within 0.5 dB of the finite-length limit and they show no visible signs of an error floor down to a BER of 10^{-7}.

From the results presented in Figs. 12 and 16, we see that puncturing has virtually no negative effect on the performance of SBC codes, and thus different approaches to achieving higher code rates can be compared on the basis of implementation complexity.

VIII. Conclusions

In this paper, we introduced a novel sliding window decoding scheme for blockwise SBC codes and proposed both uniform and nonuniform iterative decoding schedules that can significantly reduce the decoding memory and latency requirements. We used a density evolution analysis to guide the selection of the window size and the decoding schedule. We examined the BER performance and analyzed the computational complexity of blockwise SBC codes, and we compared their behavior with turbo codes under an equal latency

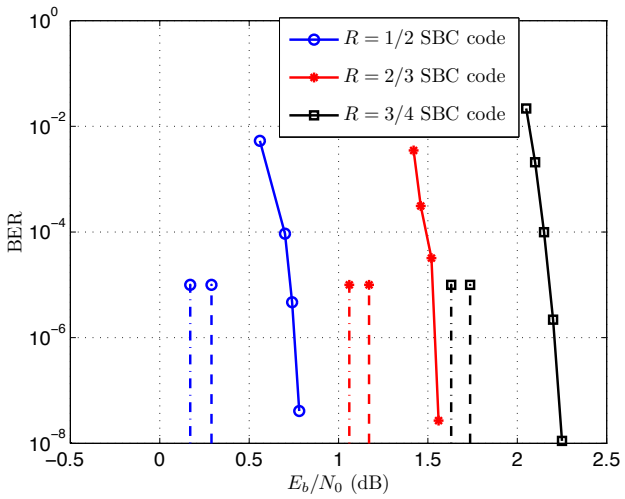


Fig. 16. Window decoding performance comparison of rate-compatible SBC codes obtained by periodic puncturing of a rate 1/2 mother code with the finite-length limit (dashed lines) and the Shannon limit (dot-dash lines). The decoding latencies of the rate-compatible SBC codes with rates $1/2$, $2/3$, and $3/4$ are $\tau = 108,000$, $81,000$, and $72,000$ bits, respectively.

constraint. Periodic puncturing was then used to achieve rate-compatible blockwise SBC codes, all of which perform within 0.5 dB of the finite-length limit. Moreover, these codes show no visible sign of an error floor down to a BER of 10^{-8}. Finally, a comparison of a blockwise SBC code, an LDPC convolutional code, and the turbo code used in the CDMA2000 standard, all with code rate $R \approx 1/2$ and the same decoding latency, showed that blockwise SBC codes can outperform both LDPC convolutional codes and turbo codes.

Based on their excellent performance in both the waterfall and error floor regions, their robust behavior with puncturing, and their ability to employ a low-complexity soft decision iterative decoding algorithm at high code rates, blockwise SBC codes appear to be worthy competitors to the product-like codes that have recently been proposed for high-speed optical communication.

REFERENCES

[1] 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.

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

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

[4] G. Zémor, "On expander codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 835-837, Feb. 2001.

[5] Y. Y. Jian, H. D. Pfister, K. R. Narayanan, R. Rao and R. Mazahreh, "Iterative hard-decision decoding of braided BCH codes for high-speed optical communication," in *Proc. IEEE Global Telecommunications Conference*, Dec. 2013.

[6] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang and J. Lodge, "Staircase codes: FEC for 100 Gb/s OTN," *J. Lightwave Technol.*, vol. 30, no. 1, pp. 110-117, 2012.

[7] 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.

[8] W. Zhang, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Braided convolutional codes," in *Proc. IEEE Int. Symp. Information Theory*, Adelaide, Australia, Sep. 2005, pp. 592-596.

[9] S. Moloudi and M. Lentmaier, "Density evolution analysis of braided convolutional codes on the erasure channel," in *Proc. IEEE Int. Symp. Information Theory*, Honolulu, HI, USA, July 2014, pp. 2609-2613.

[10] M. Lentmaier, S. Moloudi, and A. Graell i Amat, "Braided convolutional codes - a class of spatially coupled turbo-like codes," in *Proc. Int. Conf. Signal Processing and Communications*, Bangalore, India, July 2014, pp. 1-5.

[11] S. Moloudi, M. Lentmaier, and A. Graell i Amat, "Spatially coupled turbo-like codes," https://arxiv.org/abs/1604.07315.

[12] M. Zhu, D. G. M. Mitchell, M. Lentmaier, D. J. Costello, Jr., and B. Bai, "Window decoding of braided convolutional codes," in *Proc. IEEE Information Theory Workshop - Fall (ITW)*, Jeju Island, Korea, Oct. 2015, pp. 143-147.

[13] M. Lentmaier, A. Sridharan, K. S. Zigangirov, and D. J. Costello, Jr., "Terminated LDPC convolutional codes with thresholds close to capcacity," in *Proc. IEEE Int. Symp. Information Theory*, Adelaide, Australia, Sep. 2005, pp. 1372-1376.

[14] 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.

[15] 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.

[16] N. ul Hassan, A. E. Pusane, M. Lentmaier, G. P. Fettweis, and D. J. Costello, "Non-uniform windowed decoding schedules for spatially coupled codes," in *Proc. IEEE Global Communications Conference*, Atlanta, GA, Dec. 2013, pp.1862-1867.

[17] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429-445, Mar. 1996.

[18] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *JPL TDA Progr. Rep.*, vol. 42, no.127, pp. 1-20, Nov. 1996.

[19] B. M. Kurkoski, P. H. Siegel, and J. K. Wolf, "Exact probability of erasure and a decoding algorithm for convolutional codes on the binary erasure channel," in *Proc. IEEE Global Telecommunications Conference*, Dec. 2003, pp. 1741-1745.

[20] S. Lin, and D. J. Costello, Jr., *Error Control Coding (Second Edition)*. Pearson Prentice-Hall, 2004.

[21] TIA/EIA/IS-2000.2, "Physical Layer Standard for CDMA2000 Spread Spectrum Systems," *Telecommunication Industry Association*, 1999.

[22] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Comm. Letters*, vol. 16, no. 12, pp. 2044-2047, Dec. 2012.

[23] Y. Polyanskiy, H. Vincent Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307-2359, May 2010.

[24] D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, Jr., "Spatially coupled LDPC codes constructed from protographs," *IEEE Trans. Inf. Theory*, vol. 61, no. 9, pp. 4866-4889, Sept. 2015.

Min Zhu (S'15-M'17) received the B.S., the M.S. and the Ph. D. degrees in communication and information system from Xidian University, China, in 2006, 2009, and 2016 respectively. From Sept. 2014 to Sept. 2015, she was with the Department of Electronic Engineering, University of Notre Dame, IN, as a visiting Ph. D student. She is currently with State Key Lab. of ISN, Xidian University. Her research interests include channel coding and their applications to communication systems.

David G. M. Mitchell received the Ph.D. degree in Electrical Engineering from the University of Edinburgh, United Kingdom, in 2009. Since 2015, he has been an Assistant Professor in the Klipsch School of Electrical and Computer Engineering at the New Mexico State University, USA. He previously held Visiting Assistant Professor and Post-Doctoral Research Associate positions in the Department of Electrical Engineering at the University of Notre Dame, USA. He is a Senior Member of the IEEE and his research interests are in the area of digital communications, with emphasis on error control coding and information theory.

Michael Lentmaier (S'98-M'03-SM'11) received the Dipl.-Ing. degree in electrical engineering from University of Ulm, Germany in 1998, and the Ph.D. degree in telecommunication theory from Lund University, Sweden in 2003. He then worked as a Post-Doctoral Research Associate at University of Notre Dame, Indiana and at University of Ulm. From 2005 to 2007 he was with the Institute of Communications and Navigation of the German Aerospace Center (DLR) in Oberpfaffenhofen, where he worked on signal processing techniques in satellite navigation receivers. From 2008 to 2012 he was a senior researcher and lecturer at the Vodafone Chair Mobile Communications Systems at TU Dresden, where he was heading the Algorithms and Coding research group. Since January 2013 he is an Associate Professor at the Department of Electrical and Information Technology at Lund University. His research interests include design and analysis of coding systems, graph based iterative algorithms and Bayesian methods applied to decoding, detection and estimation in communication systems. He is a senior member of the IEEE and served as an editor for IEEE Communications Letters (2010-2013), IEEE Transactions on Communications (2014-2017), and IEEE Transactions on Information Theory (since April 2017). He was awarded the Communications Society & Information Theory Society Joint Paper Award (2012) for his paper Iterative Decoding Threshold Analysis for LDPC Convolutional Codes.

Daniel J. Costello, Jr. received the M.S. and Ph.D. degrees in Electrical Engineering from the University of Notre Dame, Notre Dame, IN, in 1966 and 1969, respectively. Dr. Costello joined the faculty of the Illinois Institute of Technology, Chicago, IL, in 1969. In 1985 he became Professor of Electrical Engineering at the University of Notre Dame, Notre Dame, IN, and from 1989 to 1998 served as Chair of the Department of Electrical Engineering. In 2000, he was named the Leonard Bettex Professor of Electrical Engineering at Notre Dame, and in 2009 he became Bettex Professor Emeritus.

Dr. Costello has been a member of IEEE since 1969 and was elected Fellow in 1985. In 2009, he was co-recipient of the IEEE Donald G. Fink Prize Paper Award, which recognizes an outstanding survey, review, or tutorial paper in any IEEE publication issued during the previous calendar year. In 2012, he was a co-recipient of the joint IEEE Information Theory Society/Communications Society Prize Paper Award, which recognizes an outstanding research paper in the IT or COM Transactions during the previous two calendar years. In 2013, he received the Aaron D. Wyner Distinguished Service Award from the IEEE Information Theory Society, which recognizes outstanding leadership in and long standing exceptional service to the Information Theory community. In 2015 he received the IEEE Leon K. Kirchner Graduate Teaching Award, which recognizes inspirational teaching of graduate students in the IEEE fields of interest.

Dr. Costello's research interests are in digital communications, with special emphasis on error control coding and coded modulation. He has numerous technical publications in his field, and in 1983 he co-authored a textbook entitled "Error Control Coding: Fundamentals and Applications", the 2nd edition of which was published in 2004.

Baoming Bai (S'98-M'00) received the B.S. degree from the Northwest Telecommunications Engineering Institute, China, in 1987, and the M.S. and Ph.D. degrees in communication engineering from Xidian University, China, in 1990 and 2000, respectively. From 2000 to 2003, he was a Senior Research Assistant in the Department of Electronic Engineering, City University of Hong Kong. Since April 2003, he has been with the State Key Laboratory of Integrated Services Networks (ISN), School of Telecommunication Engineering, Xidian University, China, where he is currently a Professor. In 2005, he was with the University of California, Davis, as a visiting scholar. His research interests include information theory and channel coding, wireless communication, and quantum communication.