Braided Convolutional Codes: A New Class of Turbo-Like Codes

Zhang, Wei; Lentmaier, Michael; Zigangirov, Kamil; Costello, Daniel J., Jr.

Published in:
IEEE Transactions on Information Theory

DOI:
10.1109/TIT.2009.2034784

2010

Braided Convolutional Codes: A New Class of Turbo-Like Codes

Wei Zhang, *Member, IEEE*, Michael Lentmaier, *Member, IEEE*, Kamil Sh. Zigangirov, *Fellow, IEEE*, and Daniel J. Costello, Jr., *Life Fellow, IEEE*

***Abstract*—We present a new class of iteratively decodable turbo-like codes, called braided convolutional codes. Constructions and encoding procedures for tightly and sparsely braided convolutional codes are introduced. Sparsely braided codes exhibit good convergence behavior with iterative decoding, and a statistical analysis using Markov permutors shows that the free distance of these codes grows linearly with constraint length, i.e., they are asymptotically good.**

***Index Terms*—braided convolutional codes, turbo-like codes, codes on graphs, iterative decoding, convolutional permutor, free distance.**

I. Introduction

Braided block codes (BBC's) [1] were first introduced in [2] [3]. These codes can be viewed as a sliding version of product codes [4] or expander codes [5] [6]. In braided codes, information symbols are checked by two component encoders, and the parity symbols of one component encoder are used as inputs to the other component encoder. The connections between the two component encoders are defined by the positions where information symbols and parity symbols are stored in a two-dimensional array. Braided codes form a class of continuously decodable codes defined on graphs [2], and thus iterative decoding can be employed. Owing to the continuously decodable property of these codes, the decoder can be implemented using a highly efficient pipeline structure. Therefore braided codes are well suited for high speed continuous data transmission.

In [3], short block codes such as Hamming codes were employed as component codes. Two families of BBC's were proposed based on the density of the storage array, i.e., tightly braided block codes (TBBC's) and sparsely braided block codes (SBBC's), and it was shown that iterative decoding performance is greatly improved with SBBC's.

In this paper, we study a new class of braided codes, *braided convolutional codes* (BCC's), first introduced in [7]. In contrast to BBC's, which are described in detail in [1], we use convolutional codes as component codes. Convolutional permutors, an important ingredient of BCC's, are introduced in Section II, and code constructions are described in Section III. Analogous to BBC's, a *tightly braided convolutional code* (TBCC) results when a dense array is used to store the information and parity symbols. *Sparsely braided convolutional codes* (SBCC's) are then proposed to overcome the short cycles in the Tanner graph representation [8] of TBCC's. The storage array of SBCC's has a lower density, resulting in improved iterative decoding performance. In Section IV a syndrome former matrix is defined, and SBCC's are shown to be a type of low density parity check (LDPC) convolutional code. Then in Section V a pipeline decoder architecture for high speed continuous data transmission is presented. In Section VI, a blockwise version of BCC's is proposed for applications involving packetized data. The performance of rate $R = 1/3$ SBCC's is then evaluated by computer simulation in Section VII. By means of a Markov permutor analysis [9], a numerical method is developed in Section VIII to compute a lower bound on free distance for the ensemble of BCC's. The free distance bound shows linear growth in free distance as a function of constraint length. This implies that BCC's, in contrast to turbo codes or serially concatenated codes, are asymptotically good in terms of distance growth. Finally, we present some conclusions in Section IX.

Manuscript received May 31, 2006; revised January 30, 2009.

This work was supported in part by NSF Grant CCR02-05310, Army grant DAAD16-02-C-0057, and NASA grant NNG05GH73G.

The material of this paper was presented in part at the 2005 IEEE International Symposium on Information Theory, Adelaide, Australia, September 2005.

W. Zhang was with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA. He is now with QUALCOMM Incorporated, San Diego, CA (e-mail:wzhang@qualcomm.com).

M. Lentmaier was with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA. He is now with Vodafone Chair Mobile Communications Systems, Dresden University of Technology, 01062 Dresden, Germany (e-mail: Michael.Lentmaier@ifn.et.tu-dresden.de).

K. Sh. Zigangirov is with University of Notre Dame, Notre Dame, IN 46556 USA, Lund University, Lund, Sweden, and the Institute for Problems of Information Transmission, Moscow, Russia (e-mail: kzigangi@nd.edu).

D. J. Costello, Jr. is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: Daniel.J.Costello.2@nd.edu).

Communicated by T. Richardson, Associate Editor for Coding Theory.

II. Convolutional Permutors

An essential part of the encoder for BCC's is a convolutional permutor (also called a convolutional scrambler [10]). In this section, we briefly review the basic theory of multiple convolutional permutors given in [1].

A *symmetric multiple convolutional permutor* (MCP) of *multiplicity* k can be described by a semi-infinite matrix $\mathbf{P} = (p_{t,t'})$, $t, t' \in \mathbb{Z}^+$, which has k ones in each row and in each column starting from the Δth column. The other entries are zeros. The matrix $\mathbf{P}$ also satisfies the causality condition, i.e.,

$$p_{t,t'} = 0, \ t' < t. \tag{1}$$

We use the following representation for P:

$$\mathbf{P} = \begin{pmatrix} p_{0,\delta} & p_{0,\delta+1} & \cdots & p_{0,\Delta} & & & \\ & \ddots & & & \ddots & & \\ & & p_{t,t+\delta} & p_{t,t+\delta+1} & \cdots & p_{t,t+\Delta} & \\ & & & \ddots & & & \ddots \end{pmatrix}, \tag{2}$$

and we assume that $p_{t,t+\delta} = 1$ for at least one value of t and $p_{t,t+\Delta} = 1$ for at least one value of t. The parameter $\delta \geq 0$ is called the *minimal permutor delay* and $\Delta \geq 0$ is called the *maximal permutor delay*. As in convolutional coding, we call the maximal delay the *memory* m of the permutor, i.e., $m = \Delta$. The value $w = \Delta - \delta + 1$ is called the *permutor width*. A *single convolutional permutor* has multiplicity $k = 1$. If $w = 1$ and $p_{t,t} = 1\ \forall t$, a single permutor is the *identity permutor*. If $p_{t,t+\delta} = 1\ \forall t$, a single permutor is the *delay permutor* with delay δ. If a multiple permutor is described by the matrix $\mathbf{P}$, the inverse permutor is described by the transpose matrix $[\mathbf{P}]^{\mathrm{T}}$.

With this matrix representation, we can describe a single convolutional permutor as follows. Let $\mathbf{x} = (x_0, x_1, \ldots)$ be the input sequence to the permutor. Then the output sequence $\mathbf{y} = (y_0, y_1, \ldots)$ is given by

$$\mathbf{y} = \mathbf{x}\mathbf{P}. \tag{3}$$

In this way, the mapping between the input and output is defined as $y_t = x_{t'}$, where t' is determined by the permutation function $\mathrm{f}_{\mathbf{P}}(\cdot)$ associated with $\mathbf{P}$, i.e.,

$$t' = \mathrm{f}_{\mathbf{P}}(t). \tag{4}$$

Equation (3) describes the operation of a single convolutional permutor, but the operation of a multiple ($k > 1$) convolutional permutor can't be described as the multiplication of a vector by a matrix.

In the case of a multiple permutor, the 1 entries in the matrix $\mathbf{P}$ represent memory units that can store an input symbol. The input sequence $\mathbf{X}$ entering the MCP is divided into k-tuples, i.e., $\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots)$, where $\mathbf{x}_t = (x_{t,1}, x_{t,2}, \ldots, x_{t,k})^{\mathrm{T}}$. The blocks $\mathbf{x}_t$, $t = 0, 1, \ldots$, are written to the memory units row by row. The output sequence $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_t, \ldots)$, where $\mathbf{y}_t = (y_{t,1}, y_{t,2}, \ldots, y_{t,k})^{\mathrm{T}}$, is read out column wise. Since there are the same number of ones in each row and column, every input symbol occurs once and only once in the output sequence.

To describe the operation of a multiple convolutional permutor, a *matrix permutation operator* or *permutation tensor* $\mathbf{P}$ can be introduced. (Refer to [1] for details.) Similar to a single convolutional permutor, we define the mapping between inputs and outputs as

$$y_{t,i} = x_{t',i'},\ 1 \leq i \leq k, \tag{5}$$

where t' and i' are determined by the permutation functions $\mathrm{f}_{\mathbf{P}}(\cdot,\cdot)$ and $\mathrm{g}_{\mathbf{P}}(\cdot,\cdot)$ associated with the permutation operator $\mathbf{P}$ as follows

$$t' = \mathrm{f}_{\mathbf{P}}(t, i), \tag{6}$$

$$i' = \mathrm{g}_{\mathbf{P}}(t, i). \tag{7}$$

These permutation functions are stored in a ROM for implementation.

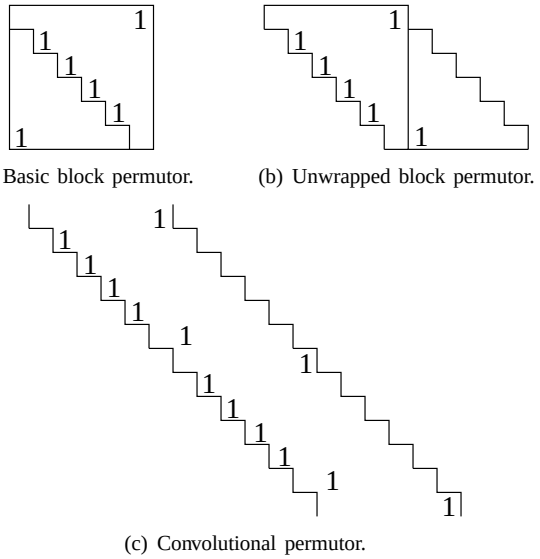
To reduce the storage space required by the permutation functions, periodic permutors are assumed. In this case,

$$p_{t,t'} = p_{t+T,t'+T},\ \forall\, t,\ t'. \tag{8}$$

The minimal T for which (8) is satisfied is called the *period* of a periodic convolutional permutor.

In [11], [1], a method was proposed to construct periodic multiple convolutional permutors from multiple block permutors. A $T \times T$ block permutor of multiplicity k is described by a $T \times T$ square matrix having k ones in each row and each column. A periodic multiple convolutional permutor with period T is then constructed from the *basic multiple block permutor* of size $T \times T$ and multiplicity k using the so-called *unwrapping procedure* [1].

Example 1: The construction of a single convolutional permutor with period $T = 6$, minimal delay $\delta = 0$, and maximal delay $\Delta = 5$, from a 6×6 basic block permutor of multiplicity $k = 1$ is illustrated in Figure 1. First divide the 6×6 permutation matrix describing the basic block permutor below the diagonal as shown in Figure 1(a), then unwrap the lower part of the matrix as shown in Figure 1(b), and finally replicate the unwrapped matrix diagonally as shown in Figure 1(c). ■



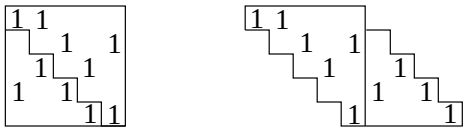
(a) Basic block permutor. (b) Unwrapped block permutor.

(c) Convolutional permutor.

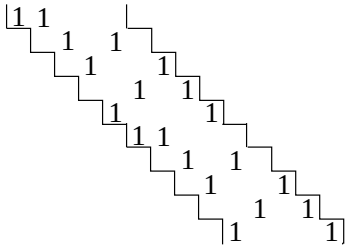
Fig. 1. Construction of a single periodic convolutional permutor.

The convolutional permutor introduced in Example 1 is a single periodic convolutional permutor. Single convolutional permutors are used in this paper to describe rate $R = 1/3$ BCC's. An example of an MCP with multiplicity $k = 2$ and period $T = 5$ constructed using the unwrapping procedure is shown in Figure 2.

From the unwrapping procedure, we see that a single periodic convolutional permutor constructed as described above may not always have minimal delay $\delta = 0$ and maximal delay (memory) $\Delta = m = T - 1$. In other words, its width is not necessarily T. However, as shown in [1], if a block permutor of multiplicity k is chosen randomly, then with probability



(a) Multiple block permutor. (b) Unwrapped block permutor.



(c) Multiple convolutional permutor.

Fig. 2. Construction of a multiple periodic convolutional permutor.

$\approx 1-(1/e)^k$ the maximal delay (memory) of the unwrapped multiple convolutional permutor of multiplicity k equals $T-1$.

The memory m is an important parameter characterizing the behavior of a convolutional permutor. Another important parameter is its *overall constraint length* M. For a given t, we introduce the set

$$\mathcal{P}_t = \{p_{i,j} : i \leq t, j > t\} \ , t \in \mathbb{Z}^+ \ . \tag{9}$$

The *overall constraint length* of the convolutional permutor is then defined by

$$M = w_{_H}(\mathcal{P}_t), \tag{10}$$

where $w_{_H}(\mathcal{P}_t)$ is the Hamming weight of the set $\mathcal{P}_t$. It follows that M is equal to the maximum number of symbols that is stored in a realization of the permutor at any time, analogous to the definition of overall constraint length for convolutional codes [10], [12]. For single convolutional permutors, since each row and column of $\mathbf{P}$ have only a single "1", the weight of $\mathcal{P}_t$ does not depend on the time index t, and we can omit t in defining M. Thus the overall constraint length is independent of t for single convolutional permutors.

Example 2: Figure 3 illustrates a single convolutional permutor with the same parameters, $T = 6$, $\delta = 0$, and $\Delta = 5$, as the convolutional permutor shown in Figure 1(c). Its overall constraint length is $M = 4$. By contrast, the convolutional permutor in Figure 1(c) has overall constraint length $M = 1$. ■

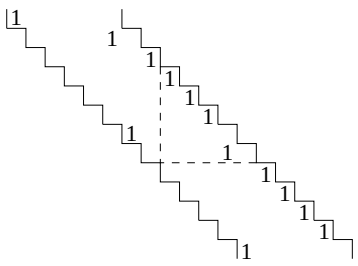


Fig. 3. A single periodic convolutional permutor with $T = 6$, $\delta = 0$, $\Delta = 5$, and $M = 4$.

For $w > 1$, the overall constraint length of a single convolutional permutor must satisfy

$$0 \leq M \leq T-2. \tag{11}$$

The single convolutional permutors for the BCC's considered in this paper were constructed from a basic block permutor (permutation matrix) chosen randomly, assuming that all $T!$ possible permutation matrices of size $T \times T$ are equiprobable. The delays of the corresponding convolutional permutors then satisfy $0 \leq \delta \leq \Delta \leq T-1$, and we note that the identity permutor has parameters $T = 1$ and $\delta = \Delta = M = 0$. Multiple convolutional permutors of multiplicity k for BCC's can be constructed from sets of k^2 permutation matrices by using the operations of row- and column-interleaving and unwrapping (see [1] for details).

Convolutional permutors constructed from $T \times T$ block permutors cannot have period larger than T. Their periods can be T, $T/2$, $T/3$, ..., and so on. If the period is $T/2$ (T even), then the $(T/2+i)$-th row of the basic block permutor is a cyclic shift of the i-th row, for $1 \leq i \leq T/2$. Similar arguments are valid for periods of $T/3$, $T/4$, ..., and so on. The probability that the cyclic shift condition is satisfied goes to zero as $T \to \infty$ for randomly chosen permutors.

An MCP of multiplicity k constructed from a $T \times T$ block permutor is called *typical* [1] if it has period T, maximal delay (memory) $\Delta = T-1$, and overall constraint length

$$M = k(T-1)/2 \ . \tag{12}$$

Shifting a *typical* MCP of multiplicity k by $a > 0$ symbols, i.e., $p_{t,t'} \to p_{t+a,t'+a}$, we obtain an MCP with additional delay a. For this permutor, the minimal delay is $\delta + a$, the maximal delay is $\Delta + a$, and the overall constraint length is

$$M = ka + k(T-1)/2. \tag{13}$$

In general, a single convolutional permutor with maximal delay Δ can be implemented with a shift register of length Δ. The permutation function $\mathrm{f}_{\mathbf{P}}(\cdot)$ associated with the permutor is stored in a controller to indicate the output indices of the register stages. At each time unit, the permutor selects an output from one of the stored symbols according to the permutation function. Then it deletes the right most symbol and shifts all other symbols one stage to the right. The new input symbol is placed into the left most position.

III. Construction of Braided Convolutional Codes

In this section, we describe the construction of BCC's. In general, braided codes, including BBC's [2] [3] and BCC's, represent a sliding version of classic product codes [4]. As illustrated in Figure 4, product codes are constructed based on a rectangular array that stores the coded symbols. The $k_1 k_2$ information (systematic) symbols are located in the upper-left corner of the array. The symbols in each row form a codeword of a *horizontal* component code $\mathcal{C}_1(n_1, k_1)$. Meanwhile, the symbols of each column form a codeword of a *vertical* component code $\mathcal{C}_2(n_2, k_2)$. In contrast, braided codes are constructed on an infinite two-dimensional array. Furthermore,

the horizontal and vertical encoders are linked through parity feedback. In this manner, the systematic and parity symbols are "braided" together.

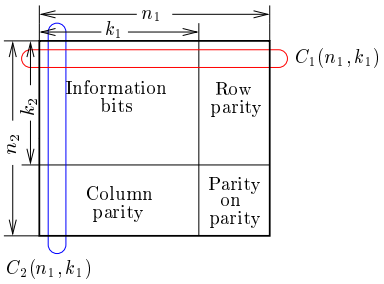


Fig. 4. An $(n_1 n_2, k_1 k_2)$ product code.

A. Rate $R = 1/3$ Braided Convolutional Codes

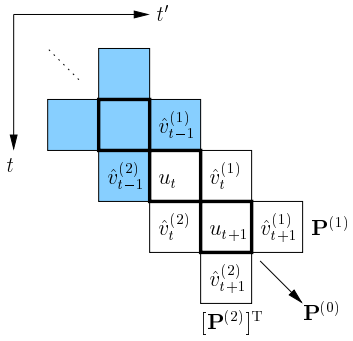


Fig. 5. Array representation of a rate $R = 1/3$ TBCC.

Depending on the density of the array, we can distinguish two types of BCC's – TBCC's and SBCC's. An example of a rate $R = 1/3$ TBCC is illustrated in Figure 5. Similar to turbo codes, recursive systematic convolutional (RSC) encoders with rate $R = 2/3$ are used as horizontal and vertical component encoders. The array consists of three diagonal ribbons, each of width one symbol. Each entry in the array is characterized by a pair of position indices (t, t'): the vertical position t and the horizontal position t', $t - 1 \leq t' \leq t + 1$. The information symbols u_t are placed in the central ribbon with position indices (t, t'), where $t = t'$, corresponding to an identity permutor $\mathbf{P}^{(0)}$. The parity symbols $\hat{v}_t^{(1)}$ of the horizontal encoder (encoder 1) are stored in the upper ribbon with position indices $(t, t+1)$. We may consider that the upper ribbon is described by a delay-1 permutor and is denoted $\mathbf{P}^{(1)}$. The parity symbols $\hat{v}_t^{(2)}$ of the vertical encoder (encoder 2) are stored in the lower ribbon with position indices $(t+1, t)$. The lower ribbon corresponds to the transpose of a delay-1 permutor and is denoted $[\mathbf{P}^{(2)}]^{\mathrm{T}}$. The dark entries in the array indicate the previous inputs and outputs of the encoders that are known at time t. Note that at time 0, when the first information symbol arrives, the previous parity symbols are assumed to be 0, i.e., $\hat{v}_t^{(1)}$ and $\hat{v}_t^{(2)}$ are zeros for $t < 0$. At time t, the horizontal encoder encodes the current information symbol u_t and its left neighbor $\hat{v}_{t-1}^{(2)}$. The output symbol $\hat{v}_t^{(1)}$ depends on $\hat{v}_{t-1}^{(2)}$, u_t, and the convolutional encoder state. The vertical encoder performs its encoding analogously. So the t-th row of the array contains $\hat{v}_{t-1}^{(2)}$, u_t, and $\hat{v}_t^{(1)}$, and the t-th column of the array contains $\hat{v}_{t-1}^{(1)}$, u_t, and $\hat{v}_t^{(2)}$. The encoding procedure continues in this fashion as the horizontal and vertical encoders slide down and to the right along the diagonal. The code sequence of the horizontal encoder is $\mathbf{v}^{(1)} = (\mathbf{v}_0^{(1)}, \mathbf{v}_1^{(1)}, \ldots, \mathbf{v}_t^{(1)}, \ldots)$, where $\mathbf{v}_t^{(1)} = (v_{t,1}^{(1)}, v_{t,2}^{(1)}, v_{t,3}^{(1)})$, $v_{t,1}^{(1)} = u_t$, $v_{t,2}^{(1)} = \hat{v}_{t-1}^{(2)}$, and $v_{t,3}^{(1)} = \hat{v}_t^{(1)}$. The code sequence of the vertical encoder is $\mathbf{v}^{(2)} = (\mathbf{v}_0^{(2)}, \mathbf{v}_1^{(2)}, \ldots, \mathbf{v}_t^{(2)}, \ldots)$, where $\mathbf{v}_t^{(2)} = (v_{t,1}^{(2)}, v_{t,2}^{(2)}, v_{t,3}^{(2)})$, $v_{t,1}^{(2)} = u_t$, $v_{t,2}^{(2)} = \hat{v}_{t-1}^{(1)}$, and $v_{t,3}^{(2)} = \hat{v}_t^{(2)}$. The code sequence transmitted over the channel is $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t = (v_{t,1}, v_{t,2}, v_{t,3})$, and

$$v_{t,i} = \begin{cases} u_t & , \ i = 1 \\ \hat{v}_t^{(1)} & , \ i = 2 \\ \hat{v}_t^{(2)} & , \ i = 3 \end{cases}. \tag{14}$$

The rate of the TBCC is $R = 1/3$. During the encoding process, two previously encoded parity bits are stored in the array, and thus the overall constraint length is $M = 2$.

Short cycles are generated in the Tanner graph of TBCC's due to their dense array structure. Thus iterative decoding performance can be improved if the cycle length is increased. This motivates the construction of SBCC's, in which information symbols and parity symbols are spread out in a sparse array. An example of the array representation of a rate $R = 1/3$ SBCC is illustrated in Figure 6. Each row and column of the array contains one information symbol, one parity symbol from the vertical encoder, and one parity symbol from the horizontal encoder. Analogous to TBCC's, the sparse array retains the three-ribbon structure and three corresponding convolutional permutors. We assume that the permutors $\mathbf{P}^{(j)} = (p_{i,k}^{(j)})$ are periodic with periods T_j, $j = \{0, 1, 2\}$, and that they are constructed using the unwrapping procedure described in Section II, with the width of each ribbon equal to the period of the corresponding permutor. Thus the widths of the central, upper, and lower ribbons are T_0, T_1, and T_2, respectively.

All the entries in the array are again indexed by coordinates (t, t'), where t and t' represent the times of the horizontal and vertical encodings, respectively, as shown in Figure 6. The information symbols u_t are placed in the central ribbon. The structure of the central ribbon is defined by the permutor $\mathbf{P}^{(0)}$. If $p_{t,t'}^{(0)} = 1$, then the t-th input symbol u_t of the encoder is placed in the array entry with index (t, t'). This means that u_t enters the horizontal encoder at time t, and the permuted symbol $\tilde{u}_{t'}$ enters the vertical encoder at time t'. Based on the analysis in Section II, a *typical* permutor $\mathbf{P}^{(0)}$ has an overall constraint length of $M_0 = (T_0 - 1)/2$. The parity symbols $\hat{v}_t^{(1)}$ of the horizontal encoder are placed in the t-th row of the upper ribbon. The structure of the upper ribbon is defined by permutor $\mathbf{P}^{(1)}$. To match the ribbon structure of the array, this permutor has an additional delay of T_0 symbols, and its overall constraint length is $M_1 = T_0 + (T_1 - 1)/2$. If $p_{t,t'}^{(1)} = 1$, then the parity symbol $\hat{v}_t^{(1)}$ is placed in the position with index (t, t'). Since $p_{t,t'}^{(1)} = 0$ for $t > t'$, the permuted parity

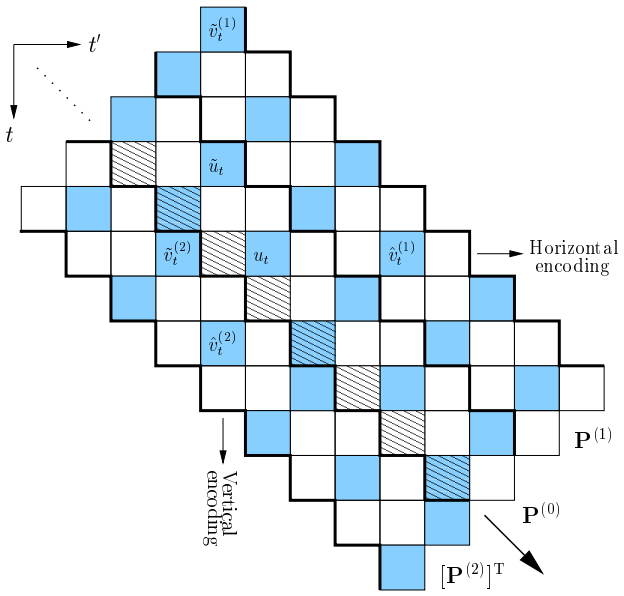


Fig. 6. Array representation for SBCC's.

symbol $\tilde{v}_{t'}^{(1)}$ will enter the vertical encoder at time t' when it leaves permutor $\mathbf{P}^{(1)}$. The parity symbols $\hat{v}_t^{(2)}$ of the vertical encoder are placed in the t-th column of the lower ribbon, whose structure depends on permutor $\mathbf{P}^{(2)}$. To match the array structure, $\mathbf{P}^{(2)}$ has minimal delay 1, maximal delay T_2, and overall constraint length $M_2 = (T_2 - 1)/2 + 1$. If $p_{t,t'}^{(2)} = 1$, then the parity symbol $\hat{v}_t^{(2)}$ is placed in the position with index (t', t). Since $p_{t,t'}^{(2)} = 0$ for $t > t'$, the permuted parity symbol $\tilde{v}_{t'}^{(2)}$ will enter the horizontal encoder at time t' when it leaves permutor $\mathbf{P}^{(2)}$.

The memory of the encoder is defined as the maximal number of time units that a symbol stays in the encoder. The *overall constraint length* M of an SBCC encoder is defined as the total number of symbols stored in the encoder. Thus, if all permutors $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ are *typical*, then

$$M = \frac{T_0 - 1}{2} + \frac{T_1 - 1}{2} + \frac{T_2 - 1}{2} + T_0 + 1. \tag{15}$$

If the permutors are all *typical* and $T_0 = T_1 = T_2 = T$, the total width of the three ribbons in a BCC is $3T$, and the total number of symbols stored in the memory of the permutors is given by

$$M = 5(T - 1)/2 + 2. \tag{16}$$

The implementation of a rate $R = 1/3$ BCC encoder is shown in Figure 7. The encoder consists of two rate $R_{cc} = 2/3$ RSC component encoders, the horizontal encoder (encoder 1) and the vertical encoder (encoder 2), and three convolutional permutors $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ are employed. The information sequence $\mathbf{u} = (u_0, u_1, \ldots, u_t, \ldots)$ enters the first input of encoder 1 directly, and the permuted information sequence $\tilde{\mathbf{u}}$ at the output of convolutional permutor $\mathbf{P}^{(0)}$ enters the first input of encoder 2. Encoder 1 generates the parity sequence $\hat{\mathbf{v}}^{(1)} = (\hat{v}_0^{(1)}, \hat{v}_1^{(1)}, \ldots, \hat{v}_t^{(1)}, \ldots)$ and encoder 2 generates the parity sequence $\hat{\mathbf{v}}^{(2)} = (\hat{v}_0^{(2)}, \hat{v}_1^{(2)}, \ldots, \hat{v}_t^{(2)}, \ldots)$.

The permuted parity sequence $\tilde{\mathbf{v}}^{(1)}$ at the output of convolutional permutor $\mathbf{P}^{(1)}$ is fed back to the second input of encoder 2, and the permuted parity sequence $\tilde{\mathbf{v}}^{(2)}$ at the output of convolutional permutor $\mathbf{P}^{(2)}$ is fed back to the second input of encoder 1. The information sequence $\mathbf{u}$ and the parity sequences $\hat{\mathbf{v}}^{(1)}$ and $\hat{\mathbf{v}}^{(2)}$ are multiplexed into the output sequence of the encoder $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t = (v_{t,1}, v_{t,2}, v_{t,3})$, and

$$v_{t,i} = \begin{cases} u_t & , \ i = 1 \\ \hat{v}_t^{(1)} & , \ i = 2 \\ \hat{v}_t^{(2)} & , \ i = 3 \end{cases}. \tag{17}$$

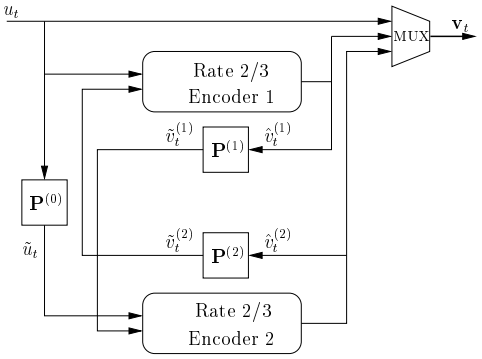


Fig. 7. Encoder for a rate $R = 1/3$ braided convolutional code.

B. Generalized Braided Convolutional Codes

Generalizing the rate $R = 1/3$ BCC's in Section III-A to other rates is straightforward. In principle, we can use different component encoders for the horizontal and vertical encodings. If we employ a rate

$$R_{cc}^{(1)} = \frac{k^{(0)} + k^{(2)}}{k^{(0)} + k^{(1)} + k^{(2)}} \tag{18}$$

horizontal encoder and a rate

$$R_{cc}^{(2)} = \frac{k^{(0)} + k^{(1)}}{k^{(0)} + k^{(1)} + k^{(2)}} \tag{19}$$

vertical encoder, where $k^{(0)}$, $k^{(1)}$, and $k^{(2)}$ are positive integers, the rate of the resulting BCC is

$$R = \frac{k^{(0)}}{k^{(0)} + k^{(1)} + k^{(2)}}. \tag{20}$$

The array representation is shown in Figure 8. The central ribbon is described by an MCP $\mathbf{P}^{(0)}$ of multiplicity $k^{(0)}$, and the upper and lower ribbons are described by MCP's $\mathbf{P}^{(1)}$ and $[\mathbf{P}^{(2)}]^{\mathrm{T}}$ of multiplicity $k^{(1)}$ and $k^{(2)}$, respectively. Horizontal and vertical encoding proceeds by row and column in the same fashion as for rate $R = 1/3$ BCC's. If the convolutional permutors are constructed from block permutors as described in Section II and they are *typical*, then the overall constraint length of the encoder is given by

$$M = \frac{k^{(0)}(T_0 - 1)}{2} + \frac{k^{(1)}(T_1 - 1)}{2} + \frac{k^{(2)}(T_2 - 1)}{2} + k^{(1)}T_0 + k^{(2)}, \tag{21}$$

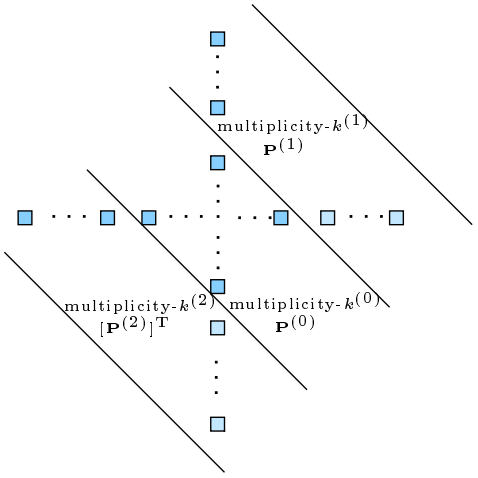


Fig. 8. Array representation of generalized BCC's.

where T_0, T_1, and T_2 are the periods of $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$, respectively.

As illustrated in Figure 9, the structure of the encoder for generalized BCC's is similar to the rate $R = 1/3$ case, except that the permutors may now be MCP's. The horizontal encoder (encoder 1) has $k^{(0)} + k^{(2)}$ inputs. At time instant t, the $k^{(0)}$-tuple information block $\mathbf{u}_t = (u_{t,1}, u_{t,2}, \ldots, u_{t,k^{(0)}})$ of the information sequence $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t, \ldots)$ enters the first $k^{(0)}$ inputs of the horizontal encoder. Meanwhile, the vertical encoder produces a block of $k^{(2)}$ parity symbols $\hat{\mathbf{v}}_t^{(2)} = (\hat{v}_{t,1}^{(2)}, \hat{v}_{t,2}^{(2)}, \ldots, \hat{v}_{t,k^{(2)}}^{(2)})$ that enters the MCP $\mathbf{P}^{(2)}$. The output $\tilde{\mathbf{v}}_t^{(2)} = (\tilde{v}_{t,1}^{(2)}, \tilde{v}_{t,2}^{(2)}, \ldots, \tilde{v}_{t,k^{(2)}}^{(2)})$ of $\mathbf{P}^{(2)}$ appears in the t-th row of the lower ribbon and provides the remaining $k^{(2)}$ inputs to the horizontal encoder. In parallel, the information sequence $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_2, \ldots, \mathbf{u}_t, \ldots)$ enters the MCP $\mathbf{P}^{(0)}$. The output sequence of $\mathbf{P}^{(0)}$ is $\tilde{\mathbf{u}} = (\tilde{\mathbf{u}}_0, \tilde{\mathbf{u}}_2, \ldots, \tilde{\mathbf{u}}_t, \ldots)$, where $\tilde{\mathbf{u}}_t = (\tilde{u}_{t,1}, \tilde{u}_{t,2}, \ldots, \tilde{u}_{t,k^{(0)}})$. The vertical encoder (encoder 2) has $k^{(0)} + k^{(1)}$ inputs. The block $\tilde{\mathbf{u}}_t$ enters the first $k^{(0)}$ inputs of vertical encoder at the time instant t. This block appears in the t-th column of the central ribbon. Meanwhile, the horizontal encoder produces a block of $k^{(1)}$ parity symbols $\hat{\mathbf{v}}_t^{(1)} = (\hat{v}_{t,1}^{(1)}, \hat{v}_{t,2}^{(1)}, \ldots, \hat{v}_{t,k^{(1)}}^{(1)})$ that enters the MCP $\mathbf{P}^{(1)}$. The output $\tilde{\mathbf{v}}_t^{(1)} = (\tilde{v}_{t,1}^{(1)}, \tilde{v}_{t,2}^{(1)}, \ldots, \tilde{v}_{t,k^{(1)}}^{(1)})$ of $\mathbf{P}^{(1)}$ appears in the t-th column of the upper ribbon and provides the remaining $k^{(1)}$ inputs to the vertical encoder. The combination of the blocks $\mathbf{u}_t$, $\hat{\mathbf{v}}_t^{(1)}$, and $\hat{\mathbf{v}}_t^{(2)}$, consisting of $k^{(0)} + k^{(1)} + k^{(2)}$ bits, forms the output code block $\mathbf{v}_t = (v_{t,1}, v_{t,2}, \ldots, v_{t,k^{(0)}+k^{(1)}+k^{(2)}})$ of the generalized BCC encoder. The multiplexing rule is defined as

$$v_{t,i} = \begin{cases} u_{t,i} & , \ 1 \le i \le k^{(0)} \\ \hat{v}_{t,i-k^{(0)}}^{(1)} & , \ 1 \le i - k^{(0)} \le k^{(1)} \\ \hat{v}_{t,i-k^{(0)}-k^{(1)}}^{(2)} & , \ 1 \le i - k^{(0)} - k^{(1)} \le k^{(2)} \end{cases} . \quad (22)$$

We can also denote the output code sequences of the horizontal ($e = 1$) and vertical ($e = 2$) encoders as $\mathbf{v}^{(e)} = (\mathbf{v}_0^{(e)}, \mathbf{v}_1^{(e)}, \ldots, \mathbf{v}_t^{(e)}, \ldots)$, where $\mathbf{v}_t^{(e)} = (v_{t,1}^{(e)}, v_{t,2}^{(e)}, \ldots, v_{t,k^{(0)}+k^{(1)}+k^{(2)}}^{(e)})$. Here, the mapping rules between the inputs and outputs of each generalized BCC component encoder can be described by

$$v_{t,i}^{(1)} = \begin{cases} u_{t,i} & , \ 1 \le i \le k^{(0)} \\ \tilde{v}_{t,i-k^{(0)}}^{(2)} & , \ 1 \le i - k^{(0)} \le k^{(2)} \\ \hat{v}_{t,i-k^{(0)}-k^{(2)}}^{(1)} & , \ 1 \le i - k^{(0)} - k^{(2)} \le k^{(1)} \end{cases} \quad (23)$$

and

$$v_{t,i}^{(2)} = \begin{cases} \tilde{u}_{t,i} & , \ 1 \le i \le k^{(0)} \\ \tilde{v}_{t,i-k^{(0)}}^{(1)} & , \ 1 \le i - k^{(0)} \le k^{(1)} \\ \hat{v}_{t,i-k^{(0)}-k^{(1)}}^{(2)} & , \ 1 \le i - k^{(0)} - k^{(1)} \le k^{(2)} \end{cases} . \quad (24)$$

At the receiver, these mapping rules determine the demultiplexing requirements of the component decoders.

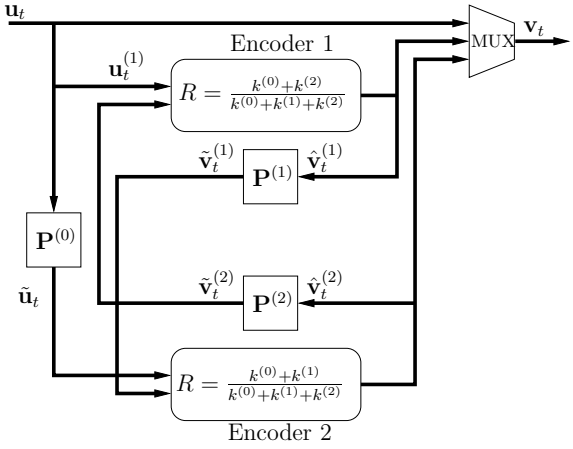


Fig. 9. Encoder for generalized BCC's.

IV. Syndrome Former Representation of Braided Convolutional Codes

In this section, we derive a canonical representation of BCC's using the syndrome former matrix. The syndrome former is useful for interpreting the structural properties of BCC's. In particular, we show that the sparsity of the permutors in the BCC encoder insures that the overall BCC syndrome former is sparse, thus making BCC's suitable for iterative decoding. We consider first some examples of the construction of syndrome formers for convolutional codes.

Example 3: Consider a rate $R_{cc} = 1/2$ RSC encoder with generator matrix

$$\mathbf{G}(D) = \begin{pmatrix} 1 & \dfrac{1}{1 + D + D^2} \end{pmatrix} . \quad (25)$$

The input sequence of the encoder is $\mathbf{u} = (u_0, u_1, \ldots, u_t, \ldots)$ and the output sequence is $\mathbf{v} = (v_0, v_1, \ldots, v_t, \ldots)$. We denote the two individual outputs of the encoder by $\mathbf{v}^{(0)} = (v_0^{(0)}, v_1^{(0)}, \ldots, v_t^{(0)}, \ldots)$ and $\mathbf{v}^{(1)} = (v_0^{(1)}, v_1^{(1)}, \ldots, v_t^{(1)}, \ldots)$. Since the encoder is systematic, $\mathbf{v}^{(0)} = \mathbf{u}$. A parity check matrix for this encoder is given by $\mathbf{H}(D) = \begin{pmatrix} 1 & \mathrm{H}^{(1)}(D) \end{pmatrix} = \begin{pmatrix} 1 & 1 + D + D^2 \end{pmatrix}$. Corresponding to $\mathrm{H}^{(1)}(D)$, we introduce

the semi-infinite matrix

$$\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}} = \begin{pmatrix} 1 & 1 & 1 & & & & \\ & 1 & 1 & 1 & & & \\ & & 1 & 1 & 1 & & \\ & & & \ddots & & & \\ & & & & & \ddots & \end{pmatrix}, \quad (26)$$

which we call the *partial syndrome former matrix*. Then the encoder's parity constraint is described by the following equation

$$\mathbf{v}^{(0)}\mathbf{I} + \mathbf{v}^{(1)}\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}, \quad (27)$$

where $\mathbf{I}$ is a semi-infinite identity matrix. ■

In order to obtain the usual description of a convolutional syndrome former, we will use the operations of *row- and column- interleaving*. These operations were introduced in [10] for two matrices and generalized in [1] for a larger number of matrices. The row-interleaving of the set of matrices $\left(\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(k)}\right)$ (see Definition 2.2 in [1]) we designate as

$$\mathbf{P} = \boxminus\left(\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(k)}\right). \quad (28)$$

Analogously, the column-interleaving of the set of matrices $\left(\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(k)}\right)$ (see Definition 2.3 in [1]) we designate as

$$\mathbf{P} = \boxplus\left(\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots, \mathbf{P}^{(k)}\right). \quad (29)$$

In Example 3, the output code sequence $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)})$, can be represented as an interleaved version of sequences $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$. If we row-interleave the matrices $\mathbf{I}$ and $\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}$, then we obtain the syndrome former $\mathbf{H}^{\mathrm{T}} = \boxminus\left(\mathbf{I}, \left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}\right)$ of the encoder in Example 3, i.e., $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$.

Example 4: Consider a rate $R_{cc} = 1/2$ RSC encoder with generator matrix

$$\mathbf{G}(D) = \begin{pmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{pmatrix}. \quad (30)$$

With input sequence $\mathbf{u} = (u_0, u_1, \ldots, u_t, \ldots)$, the output sequence $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)})$, can be represented as an interleaved version of sequences $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$, where $\mathbf{v}^{(0)} = \mathbf{u}$ and $\mathbf{v}^{(1)} = (v_0^{(1)}, v_0^{(1)}, \ldots, v_t^{(1)}, \ldots)$. A parity check matrix is given by $\mathbf{H}(D) = \left(\mathrm{H}^{(0)}(D)\ \mathrm{H}^{(1)}(D)\right) = \left(1+D^2\ \ 1+D+D^2\right)$. Then we have

$$\mathbf{v}^{(0)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}} + \mathbf{v}^{(1)}\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}, \quad (31)$$

where

$$\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}} = \begin{pmatrix} 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & 1 & 0 & 1 & & \\ & & & \ddots & & & \\ & & & & & \ddots & \end{pmatrix} \quad (32)$$

corresponds to $\mathrm{H}^{(0)}(D)$, and $\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}$ is defined in (26). The syndrome former in the conventional form is then given by $\mathbf{H}^{\mathrm{T}} = \boxminus\left(\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}, \left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}\right)$, and $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$. ■

Example 5: Consider a rate $R_{cc} = 2/3$ RSC encoder with generator matrix

$$\mathbf{G}(D) = \begin{pmatrix} 1 & 0 & \dfrac{1}{1+D+D^2} \\ 0 & 1 & \dfrac{1+D^2}{1+D+D^2} \end{pmatrix}. \quad (33)$$

The input sequences are denoted as $\mathbf{u}^{(0)} = (u_0^{(0)}, u_1^{(0)}, \ldots, u_t^{(0)}, \ldots)$ and $\mathbf{u}^{(1)} = (u_0^{(1)}, u_1^{(1)}, \ldots, u_t^{(1)}, \ldots)$. The output sequence is $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)}, v_t^{(2)})$. Since the encoder is systematic, $\mathbf{v}^{(0)} = (v_0^{(0)}, v_1^{(0)}, \ldots, v_t^{(0)}, \ldots) = \mathbf{u}^{(0)}$, $\mathbf{v}^{(1)} = (v_0^{(1)}, v_1^{(1)}, \ldots, v_t^{(1)}, \ldots) = \mathbf{u}^{(1)}$, and $\mathbf{v}^{(2)} = (v_0^{(2)}, v_1^{(2)}, \ldots, v_t^{(2)}, \ldots)$ is the parity sequence. A parity check matrix is given by $\mathbf{H}(D) = \left(1\ \mathrm{H}^{(0)}(D)\ \mathrm{H}^{(1)}(D)\right) = \left(1\ \ 1+D^2\ \ 1+D+D^2\right)$. Then we have

$$\mathbf{v}^{(0)}\mathbf{I} + \mathbf{v}^{(1)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}} + \mathbf{v}^{(2)}\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}, \quad (34)$$

where $\mathbf{I}$ is an semi-infinite identity matrix and $\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}$ and $\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}$ are defined in (32) and (26), respectively. The syndrome former is then given by

$$\mathbf{H}^{\mathrm{T}} = \boxminus\left(\mathbf{I}, \left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}, \left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}\right). \quad (35)$$

■

We now describe the construction of the syndrome former for the BCC of Figure 7. For simplicity, we assume that component encoders 1 and 2 are given by the generator matrix in (33). Let $\mathbf{u} = \mathbf{v}^{(0)}$ be the information sequence and $\hat{\mathbf{v}}^{(e)} = (\hat{\mathbf{v}}_0^{(e)}, \hat{\mathbf{v}}_1^{(e)}, \ldots, \hat{\mathbf{v}}_t^{(e)}, \ldots)$, $e \in \{1, 2\}$, where $\hat{\mathbf{v}}_t^{(e)} = (\hat{v}_{t,1}^{(e)}, \hat{v}_{t,2}^{(e)}, \hat{v}_{t,3}^{(e)})$, be the output parity sequences of encoder 1 (horizontal) and encoder 2 (vertical), respectively. Then they must satisfy the following parity constraints:

$$\mathbf{v}^{(0)}\mathbf{I} + \hat{\mathbf{v}}^{(1)}\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}} + \hat{\mathbf{v}}^{(2)}\mathbf{P}^{(2)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}} = \mathbf{0}, \quad (36)$$

$$\mathbf{v}^{(0)}\mathbf{P}^{(0)} + \hat{\mathbf{v}}^{(1)}\mathbf{P}^{(1)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}} + \hat{\mathbf{v}}^{(2)}\left[\mathbf{H}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}. \quad (37)$$

Equation (36) describes the horizontal encoder. The syndrome former $\mathbf{H}_{hor}^{\mathrm{T}}$ of the horizontal encoder is

$$\mathbf{H}_{hor}^{\mathrm{T}} = \boxminus\left(\mathbf{I}, \left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}, \mathbf{P}^{(2)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}\right), \quad (38)$$

and it follows that $\mathbf{v}\mathbf{H}_{hor}^{\mathrm{T}} = \mathbf{0}$, where $\mathbf{v}$ is the output sequence of the BCC encoder shown in Figure 7. Similarly, (37) describes the vertical encoder. Its syndrome former is

$$\mathbf{H}_{ver}^{\mathrm{T}} = \boxminus\left(\mathbf{P}^{(0)}, \mathbf{P}^{(1)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}, \left[\mathbf{H}^{(1)}\right]^{\mathrm{T}}\right), \quad (39)$$

and $\mathbf{v}\mathbf{H}_{ver}^{\mathrm{T}} = \mathbf{0}$.

It follows that the syndrome former $\mathbf{H}^{\mathrm{T}}$ of the rate $R = 1/3$ BCC in Figure 7 with rate $R_{cc} = 2/3$ component encoders given by (33) is

$$\mathbf{H}^{\mathrm{T}} = \boxplus\left(\mathbf{H}_{hor}^{\mathrm{T}}, \mathbf{H}_{ver}^{\mathrm{T}},\right) \quad (40)$$

and hence $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$. Now we have a conventional representation of the syndrome former matrix. If the periods T_0, T_1, and T_2 of permutors $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ are large enough, $\mathbf{P}^{(0)}$, $\mathbf{P}^{(2)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}$, and $\mathbf{P}^{(1)}\left[\mathbf{H}^{(0)}\right]^{\mathrm{T}}$ are also sparse. Thus the syndrome former matrix $\mathbf{H}^{\mathrm{T}}$ is sparse, and the corresponding BCC can be considered as a special case of an LDPC convolutional code [11]. The syndrome former for generalized BCC's can be expressed in a similar way by making use the row and column interleaving operations.

The model we have considered so far assumes the transmission of an infinite length information sequence. Since real communication systems transmit finite length information sequences, the encoding of BCC's should be terminated so that the information bits at the end of the input sequence are adequately protected. In convolutional coding, the normal method of termination is to add a tail to the information sequence that forces the encoder to the zero state. The tail depends both on the encoder structure and the encoder state. The tail bits can be computed by a simple termination circuit if the encoder is based on a partial syndrome realization, as developed for LDPC convolutional codes in [13] and applied to BBCs in [1]. Given a syndrome former representation of a specific code, the parameters for this termination circuit can be precomputed by solving a system of linear equations.

For the turbo-like encoder structure shown in Figure 7, the state of the BCC encoder depends not only on the states of the component encoders, but also on the states of the convolutional permutors. The determination of tail bits that drive the overall encoder to the zero state is in this case not straightforward. A suboptimal but simple way of terminating such an encoder is to append a tail of zero bits to the information sequence. In this case, only the parity bits in the tail must be transmitted. For BCC's with period T convolutional permutors, a length $2T$ zero tail has been determined to be sufficient in practice. In this case, if the length of the information sequence is L for a rate $R = 1/3$ BCC, the resulting code rate of the terminated code is given by

$$R = \frac{1}{3}\frac{L}{L + 4T/3}. \tag{41}$$

V. Pipeline Decoder Architecture

A pipeline decoder architecture for LDPC convolutional codes was first proposed in [11], where the continuously decodable property of these codes was exploited to accelerate the decoding speed. By employing a number of processors equal to the number of iterations to execute the decoding algorithm in parallel, the pipeline decoder yields estimated outputs at each execution cycle after some initial decoding delay. Since BCC's are a special class of LDPC convolutional codes, they can be decoded using the pipeline architecture. In this section, we describe the pipeline structure for continuous decoding of BCC's.

Assume that the generalized BCC encoder described in Section III-B is used. The code sequence is $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$, where $\mathbf{v}_t = (v_{t,1}, v_{t,2}, \ldots, v_{t,k^{(0)}+k^{(1)}+k^{(2)}})$. After transmitting over a memoryless channel, such as an additive white Gaussian (AWGN) channel, the received sequence is $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_t, \ldots)$, where $\mathbf{r}_t = (r_{t,1}, r_{t,2}, \ldots, r_{t,k^{(0)}+k^{(1)}+k^{(2)}})$. Using the conditional probability $p(r|v)$ of receiving the signal r given the transmitted signal v, we can calculate the channel log-likelihood ratio's (LLR's) $\mathbf{l} = (\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_t, \ldots)$, where $\mathbf{l}_t = (l_{t,1}, l_{t,2}, \ldots, l_{t,k^{(0)}+k^{(1)}+k^{(2)}})$, for the coded bits:

$$l_{t,i} = \log\frac{p(r_{t,i}|v_{t,i}=0)}{p(r_{t,i}|v_{t,i}=1)},\ t \geq 0,\ 1 \leq i \leq k^{(0)} + k^{(1)} + k^{(2)}. \tag{42}$$

According to the mapping rules (23) and (24), these LLR's are demultiplexed into two streams. For component encoder e, $e \in \{1, 2\}$, the channel LLR's corresponding to the outputs $\mathbf{v}^{(e)} = (\mathbf{v}_0^{(e)}, \mathbf{v}_1^{(e)}, \ldots, \mathbf{v}_t^{(e)}, \ldots)$, where $\mathbf{v}_t^{(e)} = (v_{t,1}^{(e)}, v_{t,2}^{(e)}, \ldots, v_{t,k^{(0)}+k^{(1)}+k^{(2)}}^{(e)})$, are given by $\mathbf{l}^{(e)} = (\mathbf{l}_0^{(e)}, \mathbf{l}_1^{(e)}, \ldots, \mathbf{l}_t^{(e)}, \ldots)$, where $\mathbf{l}_t^{(e)} = (l_{t,1}^{(e)}, l_{t,2}^{(e)}, \ldots, l_{t,k^{(0)}+k^{(1)}+k^{(2)}}^{(e)})$.

Let $\mathbf{L}(0) = (\mathbf{L}_0(0), \mathbf{L}_1(0), \ldots, \mathbf{L}_t(0), \ldots)$, where $\mathbf{L}_t(0) = (L_{t,1}(0), L_{t,2}(0), \ldots, L_{t,k^{(0)}+k^{(1)}+k^{(2)}}(0))$, be the set of *apriori* LLR's for the code sequence $\mathbf{v}$. In this way, we denote the *apriori* LLR for the coded bit $v_{t,k}$ as $L_{t,k}(0)$. The *apriori* LLR's for the code sequence $\mathbf{v}$ are given by

$$L_{t,i}(0) = \begin{cases} \infty, & t < 0 \\ 0, & t \geq 0 \end{cases},\ 1 \leq i \leq k^{(0)} + k^{(1)} + k^{(2)}. \tag{43}$$

Analogously, let $\mathbf{L}^{(1)}(0)$ and $\mathbf{L}^{(2)}(0)$ be the set of *apriori* LLR's for the code sequences $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ from the horizontal and vertical encoders, respectively. Since there is a one-one mapping between the symbols of the sequences $\mathbf{v}$ and $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ according to (22), (23), and (24), we can also find the values for $\mathbf{L}^{(1)}(0)$ and $\mathbf{L}^{(2)}(0)$.

When the transmitted signals arrive at the receiver, the channel LLR's are calculated and placed into parallel buffers along with the *apriori* LLR's. The component codes are then decoded using a parallel bank of $2I$ *a posteriori* probability (APP) processors using the windowed BCJR algorithm [14] [15], where I is the number of iterations to be performed. Based on the channel LLR's $\mathbf{l}^{(1)}$ and the *apriori* LLR's $\mathbf{L}^{(1)}(0)$, the first APP processor $\mathcal{B}_1^{(1)}$ obtains the extrinsic LLR's $\mathbf{L}^{(1)}(1)$ for a window of W coded symbols of the sequence $\mathbf{v}^{(1)}$ from the horizontal encoder. Then the extrinsic LLR's $\mathbf{L}^{(1)}(1)$ are reordered to $\mathbf{L}^{(2)}(1)$ according to the order of the code sequence $\mathbf{v}^{(2)}$ of the vertical encoder, based on the mapping rules in (23) and (24). During the reordering, the extrinsic LLR's in $\mathbf{L}^{(1)}(1)$ for $\mathbf{u}_t$, $\tilde{\mathbf{v}}_t^{(2)}$, and $\hat{\mathbf{v}}_t^{(1)}$ are permuted by $\mathbf{P}^{(0)}$, $\left[\mathbf{P}^{(2)}\right]^{\mathrm{T}}$, and $\mathbf{P}^{(1)}$, respectively. $\mathbf{L}^{(2)}(1)$ is used as *apriori* LLR's for the code sequence $\mathbf{v}^{(2)}$ by the APP processor $\mathcal{B}_2^{(2)}$. In the same manner as for the first processor $\mathcal{B}_1^{(1)}$, processor $\mathcal{B}_2^{(2)}$ calculates the extrinsic LLR's $\mathbf{L}^{(2)}(2)$ for a window of W symbols of the sequence $\mathbf{v}^{(2)}$. The extrinsic LLR's $\mathbf{L}^{(2)}(2)$ are then reordered to $\mathbf{L}^{(1)}(2)$ according to the order of the code sequence $\mathbf{v}^{(1)}$ of the horizontal encoder, based on the mapping rules in (24) and (23). During the reordering, the extrinsic LLR's in $\mathbf{L}^{(2)}(2)$ for $\tilde{\mathbf{u}}_t$, $\tilde{\mathbf{v}}_t^{(1)}$, and $\hat{\mathbf{v}}_t^{(2)}$ are permuted by $\left[\mathbf{P}^{(0)}\right]^{\mathrm{T}}$, $\left[\mathbf{P}^{(1)}\right]^{\mathrm{T}}$, and $\mathbf{P}^{(2)}$, respectively. The third APP processor $\mathcal{B}_3^{(1)}$ then uses $\mathbf{L}^{(1)}(2)$ as *apriori* LLR's. The following APP processors work in a similar

fashion as described above. A pipeline decoder comprised of $2I$ APP processors to perform I iterations of decoding is shown in Figure 10. Processors $\mathcal{B}_{2j-1}^{(1)}$ and $\mathcal{B}_{2j}^{(2)}$, $1 \le j \le I$, perform horizontal and vertical component decoding, respectively. Each processor performs the windowed BCJR algorithm on a window of size W, where W should be large compared to the constraint length of the component encoder [16]. In order to avoid different processors working on overlapping sets of coded bits at the same time, a *separation delay* of τ coded symbols is imposed between adjacent processors so that the *apriori* values are updated without memory conflicts. If T is the period of all the permutors, it is sufficient to set

$$\tau = 3T. \tag{44}$$

Eventually the received sequence flows through the series of processors $\mathcal{B}_1^{(1)}$, $\mathcal{B}_2^{(2)}$, $\mathcal{B}_3^{(1)}$, ..., $\mathcal{B}_{2I}^{(2)}$, which update the *apriori* values for the coded bits $2I$ times. The last processor $\mathcal{B}_{2I}^{(2)}$ makes hard decisions for the information bits based on its output APP values. Using this pipeline structure, we can process $2I$ information symbols in parallel, thus achieving high speed decoding.

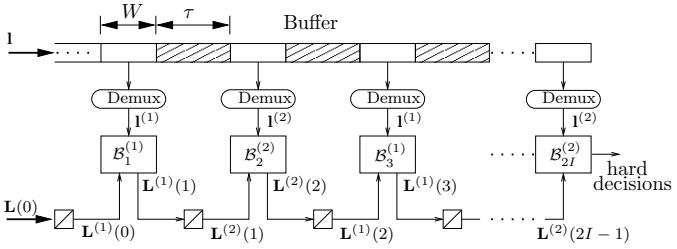


Fig. 10. Pipeline decoder for BCC's.

This procedure is similar to the decoding of turbo codes. The major difference is that the pipeline decoder uses a windowed BCJR decoder and calculates APP values for all the code symbols instead of only the information symbols. A drawback of pipeline decoding is that it has a large initial decoding delay. Only after the last processor in the pipeline has filled up does the decoder start making hard decisions on the information bits. Thus there is an initial delay (latency) of $2I(W+\tau)$ coded symbols, or about $2.5I$ times the overall constraint length of the encoder. Nevertheless, we obtain continuous decoding outputs after this initial delay.

In the next section, we consider *blockwise* BCC's. In this case, we assume that the information sequence enters the encoder in a block by block manner with a relatively large block size. This corresponds to many practical applications in which the data stream is transmitted in finite length packets. In this sense, the BCC's introduced in the previous sections are referred to as *bitwise* BCC's.

VI. Blockwise Braided Convolutional Codes

To encode a blockwise BCC the information sequence is divided into blocks of length N symbols, i.e., $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t, \ldots)$, where $\mathbf{u}_t = (u_{t,1}, u_{t,2}, \ldots, u_{t,N})$. To simplify the description, we suppose that the whole block $\mathbf{u}_t$ is sent to the encoder at time instant t. If we allow for some change of notation, a rate $R = 1/3$ blockwise BCC encoder can still be described by Figure 7. In particular, $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ now denote block permutors of size N rather than convolutional permutors. The information symbol u_t at the encoder input is replaced by the block $\mathbf{u}_t$, the parity symbol $\hat{v}_t^{(1)}$ of the horizontal encoder is replaced by the parity block $\hat{\mathbf{v}}_t^{(1)} = (\hat{v}_{t,1}^{(1)}, \hat{v}_{t,2}^{(1)}, \ldots, \hat{v}_{t,N}^{(1)})$, and the parity symbol $\hat{v}_t^{(2)}$ of the vertical encoder is replaced by the parity block $\hat{\mathbf{v}}_t^{(2)} = (\hat{v}_{t,1}^{(2)}, \hat{v}_{t,2}^{(2)}, \ldots, \hat{v}_{t,N}^{(2)})$. As component encoders we consider now rate $R = 2/3$ tail-biting convolutional encoders that start from and end in the same state. This way the trellises are decoupled between different blocks and the component decoding can be performed independently for different time instants t. A termination of the encoders to the zero state within each time instant might slightly improve the performance but at the cost of a loss in rate.

At the 0-th time instant, 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. Meanwhile, blocks $\tilde{\mathbf{v}}_{-1}^{(2)}$ and $\tilde{\mathbf{v}}_{-1}^{(1)}$, consisting of N zeros each, enter the second inputs of encoder 1 and encoder 2, respectively. Encoders 1 and 2 then generate the length N parity blocks $\hat{\mathbf{v}}_0^{(1)}$ and $\hat{\mathbf{v}}_0^{(2)}$. Blocks $\mathbf{v}_0^{(0)} = \mathbf{u}_0$, $\mathbf{v}_0^{(1)} = \hat{\mathbf{v}}_0^{(1)}$, and $\mathbf{v}_0^{(2)} = \hat{\mathbf{v}}_0^{(2)}$ are sent over the channel. At the t-th time instant, parity block $\mathbf{v}_t^{(1)}$ is calculated by encoder 1 as a function of $\mathbf{u}_t$ and $\tilde{\mathbf{v}}_t^{(1)} = \mathbf{v}_{t-1}^{(2)}\mathbf{P}^{(2)}$. 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 $\tilde{\mathbf{v}}_t^{(2)} = \mathbf{v}_{t-1}^{(1)}\mathbf{P}^{(1)}$. The blocks $\mathbf{v}_t^{(0)} = \mathbf{u}_t$, $\mathbf{v}_t^{(1)} = \hat{\mathbf{v}}_t^{(1)}$, and $\mathbf{v}_t^{(2)} = \hat{\mathbf{v}}_t^{(2)}$ are multiplexed into the code sequence

$$\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots), \tag{45}$$

where

$$\mathbf{v}_t = (v_1^{(0)}, v_1^{(1)}, v_1^{(2)}, v_2^{(0)}, v_2^{(1)}, v_2^{(2)}, \ldots, v_N^{(0)}, v_N^{(1)}, v_N^{(2)}). \tag{46}$$

In the following example, we use partial syndrome former matrices to describe the encoding process for blockwise BCC's.

Example 6: Consider the rate $R = 2/3$ encoder with generator matrix given by (33). In Examples 3–5 , (27), (31), and (34) describe the constraints implied by the encoders given in (25), (30), and (33). Suppose that the encoder in (33) is used as a tail-biting rate $R = 2/3$ encoder to encode the length N information sequences $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$. The partial syndrome formers are $N \times N$ matrices

$$\left[\tilde{\mathbf{H}}^{(0)}\right]^{\mathrm{T}} = \begin{pmatrix} 1 & 0 & 1 & & & & & \\ & 1 & 0 & 1 & & & & \\ & & & \ddots & & & & \\ & & & & & 1 & 0 & 1 \\ 1 & & & & & & 1 & 0 \\ 0 & 1 & & & & & & 1 \end{pmatrix} \tag{47}$$

and

$$\left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}} = \begin{pmatrix} 1 & 1 & 1 & & & & & \\ & 1 & 1 & 1 & & & & \\ & & & \ddots & & & & \\ & & & & 1 & 1 & 1 \\ 1 & & & & & 1 & 1 \\ 1 & 1 & & & & & 1 \end{pmatrix}, \quad (48)$$

where the overbars on $\bar{\mathbf{H}}^{(0)}$ and $\bar{\mathbf{H}}^{(1)}$ indicate the tail-biting versions of the syndrome formers. Then the code sequence $(\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$, where $\mathbf{v}^{(0)} = \mathbf{u}^{(0)}$ and $\mathbf{v}^{(1)} = \mathbf{u}^{(1)}$, satisfies the constraint

$$\mathbf{v}^{(0)} + \mathbf{v}^{(1)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}} + \mathbf{v}^{(2)}\left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}. \quad (49)$$

■

We assume that two such tail-biting convolutional encoders are used in the rate $R = 1/3$ blockwise BCC encoder. Then

$$\mathbf{v}_t^{(0)} + \mathbf{v}_{t-1}^{(2)}\mathbf{P}^{(2)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}} + \mathbf{v}_t^{(1)}\left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}, \quad (50)$$

$$\mathbf{v}_t^{(0)}\mathbf{P}^{(0)} + \mathbf{v}_{t-1}^{(1)}\mathbf{P}^{(1)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}} + \mathbf{v}_t^{(2)}\left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}} = \mathbf{0}. \quad (51)$$

Given $\mathbf{v}_t^{(0)}$, $\mathbf{v}_{t-1}^{(1)}$, and $\mathbf{v}_{t-1}^{(2)}$, (50) and (51) define the code blocks $\mathbf{v}_t^{(1)}$ and $\mathbf{v}_t^{(2)}$. These equations for $\mathbf{v}_t^{(1)}$ and $\mathbf{v}_t^{(2)}$ can be uniquely solved if and only if the matrix $\left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}}$ has a right inverse $\mathbf{G}^{(1)}$. Then

$$\mathbf{v}_t^{(1)} = \mathbf{v}_t^{(0)}\mathbf{G}^{(1)} + \mathbf{v}_{t-1}^{(2)}\mathbf{P}^{(2)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}}\mathbf{G}^{(1)}, \quad (52)$$

$$\mathbf{v}_t^{(2)} = \mathbf{v}_t^{(0)}\mathbf{P}^{(0)}\mathbf{G}^{(1)} + \mathbf{v}_{t-1}^{(1)}\mathbf{P}^{(1)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}}\mathbf{G}^{(1)}. \quad (53)$$

We can use the same techniques as in Section IV to construct the syndrome former for blockwise BCC's. The following matrices are derived from the row-interleaving operation,

$$\bar{\mathbf{H}}_{hor}^{\mathrm{T}} = \boxminus\left(\bar{\mathbf{I}}, \left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}}, \mathbf{P}^{(2)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}}\right), \quad (54)$$

$$\bar{\mathbf{H}}_{ver}^{\mathrm{T}} = \boxminus\left(\mathbf{P}^{(0)}, \mathbf{P}^{(1)}\left[\bar{\mathbf{H}}^{(0)}\right]^{\mathrm{T}}, \left[\bar{\mathbf{H}}^{(1)}\right]^{\mathrm{T}}\right), \quad (55)$$

where $\bar{\mathbf{I}}$ is the $N \times N$ identity matrix. By means of the Kronecker product [17], the syndrome former matrices for the horizontal and vertical component codes is then given by

$$\mathbf{H}_{hor}^{\mathrm{T}} = \mathbf{I} \otimes \bar{\mathbf{H}}_{hor}^{\mathrm{T}} \quad (56)$$

and

$$\mathbf{H}_{ver}^{\mathrm{T}} = \mathbf{I} \otimes \bar{\mathbf{H}}_{ver}^{\mathrm{T}}, \quad (57)$$

respectively, where $\mathbf{I}$ is the semi-infinite identity matrix so that the block matrices $\bar{\mathbf{H}}_{hor}^{\mathrm{T}}$ and $\bar{\mathbf{H}}_{ver}^{\mathrm{T}}$ are replicated infinitely along the diagonal. Corresponding to the code sequence $\mathbf{v}$ given by (45) for a rate $R = 1/3$ blockwise BCC, the syndrome former is obtained by column-interleaving the matrices $\mathbf{H}_{hor}^{\mathrm{T}}$ and $\mathbf{H}_{ver}^{\mathrm{T}}$, i.e.,

$$\mathbf{H}^{\mathrm{T}} = \boxplus\left(\mathbf{H}_{hor}^{\mathrm{T}}, \mathbf{H}_{ver}^{\mathrm{T}}\right). \quad (58)$$

If N is large, the syndrome former matrix $\mathbf{H}^{\mathrm{T}}$ of the blockwise BCC is sparse, and blockwise BCC's can be considered as special cases of LDPC convolutional codes.

Similar to bitwise BCC's, termination is used to give protection to the information blocks at the end of the input sequence for blockwise BCC's. To reduce the encoding complexity, we again use termination with a tail of all-zero blocks for blockwise BCC's. In this case, after the information blocks

$$\mathbf{u}_{[0,L-1]} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{L-1}) \quad (59)$$

enter the blockwise BCC encoder, Λ additional all-zero blocks $\mathbf{u}_L, \ldots, \mathbf{u}_{L+\Lambda-1}$ enter the encoder. Since these Λ blocks are not sent over the channel, the component encoders have, in fact, rate $R = 1/2$ instead of $R = 2/3$. The resulting rate of the BCC including the tail is

$$R = \frac{1}{3}\frac{L}{L + 2\Lambda/3}, \quad (60)$$

where a tail length $\Lambda = 2$ blocks ($2N$ bits) has been determined to be sufficient in practice.

VII. Simulation Results

In this section, the bit-error-rate (BER) performance of rate $R = 1/3$ BCC's is evaluated on an additive white Gaussian noise (AWGN) channel using computer simulation.

We consider first bitwise SBCC's with two identical rate $R_{cc} = 2/3$, memory $m_{cc} = 2$, low complexity (4-state) RSC component encoders. The generator matrix of the component encoders is given by

$$\mathbf{G}(D) = \begin{pmatrix} 1 & 0 & \dfrac{1}{1+D+D^2} \\ 0 & 1 & \dfrac{1+D^2}{1+D+D^2} \end{pmatrix}. \quad (61)$$

The three convolutional permutors $\mathbf{P}^{(0)}$, $\mathbf{P}^{(1)}$, and $\mathbf{P}^{(2)}$ used in the encoder were constructed randomly with the same period T. We assumed that transmission consists of an information sequence of length $50T$ and a tail of $2T$ zero tail bits. Thus we have a rate loss of 2.67%, i.e., the effective rate is about 0.325. In the pipeline BCJR decoder, a window length of T and $I = 100$ decoding iterations were used[1]. The results are presented in Figure 11, where we view the effect of the period T of the convolutional permutors on the error performance as a function of the signal-to-noise ratio (SNR) E_b/N_0. We see that the performance of iterative decoding improves dramatically as the permutor period increases, an effect equivalent to the "interleaver gain" of turbo codes [18]. The SBCC achieves a BER of 10^{-5} at an E_b/N_0 of 0.4dB with permutor period $T = 8000$, which is about 1dB from the capacity of the binary-input AWGN channel with code rate 0.325.

We also studied the performance of rate $R = 1/3$ blockwise BCC's. The tail-biting version of the encoder whose generator matrix is given in (61) was employed. The three block permutors used in the encoder were chosen randomly with the same size N. As above, the transmission of 50 information blocks is terminated with 2 all-zero blocks. The parameters for decoding are the same as for the bitwise SBCC case,

[1]A value of $W = T$ was chosen for convenience, but in a practical implementation, a much smaller value of W can be chosen to minimize latency [16].

with a separation delay $\tau = N$. The BER performance is shown in Figure 12, where we changed the size of the block permutors from $N = 100$ to 8000. Similar to the bitwise case, the performance of blockwise BCC's improves as we increase the size of the block permutors. Furthermore, we see that the performance of blockwise BCC's is close to the bitwise case when the block permutor size equals the convolutional permutor period. Finally, the blockwise BCC was compared to a rate $R = 1/3$ turbo code with 4-state $[1, 5/7]$ (octal format) component encoders and permutor size 8192. The turbo code exhibits an error floor at a BER of 10^{-6} and $E_b/N_0 = 0.5$dB. By contrast, the blockwise BCC's achieve a BER of 10^{-6} at $E_b/N_0 = 0.3$dB with permutor size $N = 8000$ and error floor did not show in the simulation. These results suggest that BCC's have good minimum distance properties. In the next section, we present a distance analysis for the ensemble of BCC's that confirms this observation.

Figure 13 shows the performance of the same blockwise BCC's for a continuous pipeline decoder without any termination. The corresponding density evolution threshold at 0.98dB has been estimated by tracking the probability density functions of the decoder output LLR's with Monte Carlo methods, as described in [19]. Although a different, protograph-based BCC ensemble [20] is considered in [19], the structure of the computation tree and, consequently, the asymptotic threshold are the same as for our bitwise and blockwise ensembles[2]. Already for permutor size $N = 500$ the blockwise BCC's achieve BER levels below 10^{-5} at an E_b/N_0 that is less than 0.02dB away from the estimated threshold. For larger permutors, like for BBCs [1], it can be observed that terminated blockwise BCC's have better performance and even outperform the thresholds of continuous BCC's. This again indicates that terminated convolutional codes have better thresholds than their non-terminated counterparts, as was shown in [21].

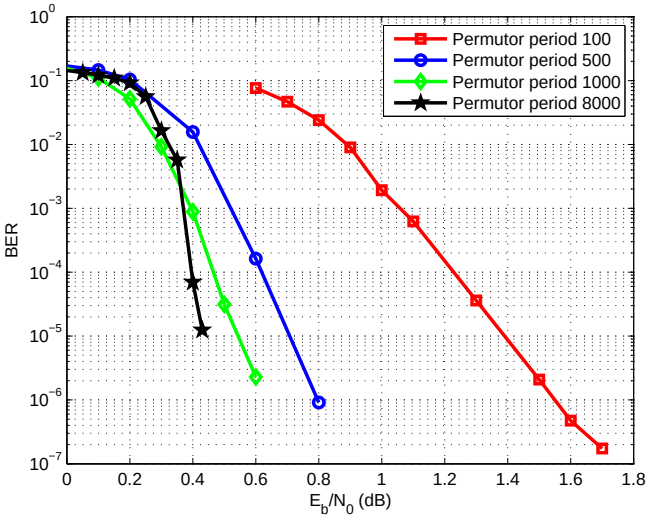


Fig. 11. Error performance of rate $R = 1/3$ terminated sparsely braided convolutional codes on an AWGN channel.

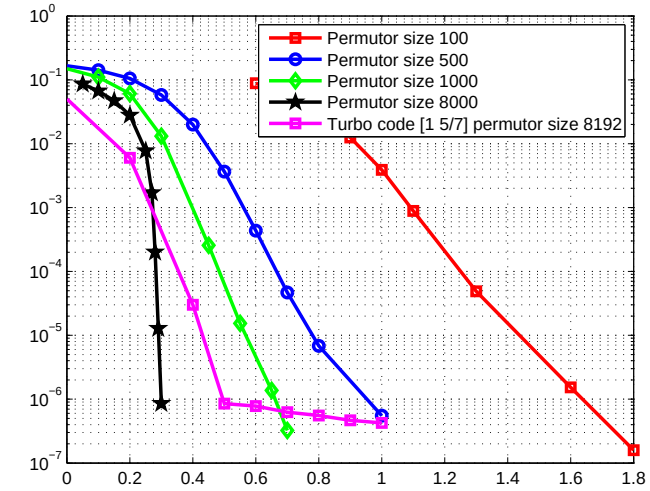


Fig. 12. Error performance of rate $R = 1/3$ terminated blockwise braided convolutional codes and turbo codes on an AWGN channel.

VIII. Statistical Analysis of Braided Convolutional Codes

One of the most important performance measures of a convolutional code is its minimum free distance d_{free}, since its large SNR performance with maximum likelihood decoding depends on d_{free}. Also, with iterative decoding, a large d_{free} protects against the appearance of an error floor at low BER's. In this section, we describe a method to compute a lower bound on the free distance of BCC's with sufficiently large overall constraint length. Using a numerical analysis for a randomized ensemble of BCC's, we obtain a lower bound on d_{free} that grows linearly with overall constraint length M as M goes to infinity.

A. Markov Permutors

In [9], a stochastic device called a *Markov permutor* was introduced to analyze the distance properties of LDPC convolutional codes. A Markov permutor is a time-varying non-periodic permutor with minimal delay $\delta = 1$ and maximal delay $\Delta = \infty$. It stores a fixed number of symbols M, i.e., *the overall constraint length of the Markov permutor* is M. To find a lower bound on free distance for the ensemble of BCC's based on Markov permutors, we define the state of the Markov permutor as the number of 1's stored in the permutor. At each time unit, the Markov permutor chooses one symbol from the stored symbols as its output symbol. The probability that a given stored symbol in the Markov permutor becomes the output symbol is $1/M$. Based on these assumptions, the probability distributions of the outputs and state transitions can be derived. In this fashion, the Markov permutor characterizes an ensemble of randomly chosen convolutional permutors with overall constraint length M.

[2]The threshold has been estimated to be at 1.10dB in [20]. This value was improved to 0.98dB by improving the resolution in the representation of the estimated probability density functions.

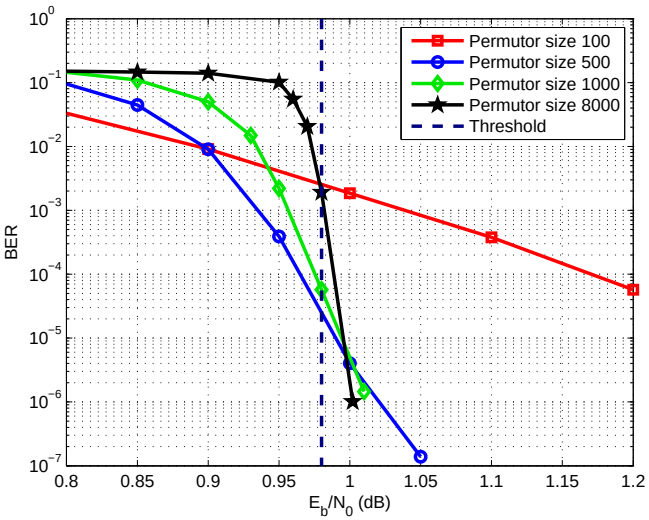


Fig. 13. Error performance of rate $R = 1/3$ continuous blockwise braided convolutional codes on an AWGN channel.

It follows that the average delay of a symbol is given by

$$\sum_{i=1}^{\infty} i \left(1 - \frac{1}{M}\right)^{i-1} \frac{1}{M} = M. \tag{62}$$

This means that a Markov permutor stores each input symbol an average of M time instants in its memory. (Note that, in contrast to fixed convolutional permutors, where a symbol cannot be held longer than the maximal delay Δ, a Markov permutor can store symbols, in principle, for an infinite time.)

Consider as an example the rate $R = 1/3$ BCC encoder in Figure 7, but replace each convolutional permutor with a Markov permutor having overall constraint length $M/3$. (The bound to be derived below can be extended to generalized BCC's in a straightforward manner.) At time instant t, $t = 0, 1, \ldots$, each permutor chooses randomly one symbol from among the $M/3$ symbols that are stored in its memory and passes this symbol to the permutor output. The permutor $\mathbf{P}^{(0)}$ replaces this symbol with a new information symbol. The permutors $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ replace their outputs with new parity symbols $v_t^{(1)}$ and $v_t^{(2)}$, respectively. The ensemble of BCC encoders with Markov permutors can be studied analytically to determine an average distance spectrum and, consequently, a lower bound on free distance for BCC's. The problem involves solving a system of recursive equations whose variables represent the path weights and the states of the permutors and the component encoders. However, this approach is quite difficult for numerical calculation. To simplify the analysis, we replace the three Markov permutors with one *multiple Markov permutor* (MMP) of overall constraint length M and multiplicity 3 (see Figure 14). By definition, an MMP of multiplicity k has k inputs and k outputs per time instant.

Initially, the MMP stores M zero symbols. At each time instant $t \geq 0$, the permutor chooses uniformly three symbols $\tilde{v}_t^{(0)}$, $\tilde{v}_t^{(1)}$, and $\tilde{v}_t^{(2)}$ from among the M symbols in its memory. As shown in Figure 14, the permutor sends this three-tuple

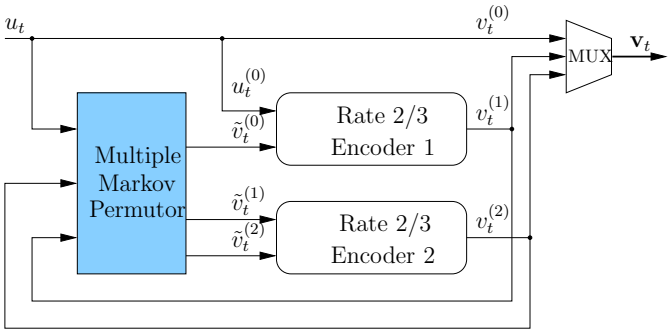


Fig. 14. Rate $R = 1/3$ BCC encoder with a multiple Markov permutor.

$\tilde{\mathbf{v}}_t = (\tilde{v}_t^{(0)}, \tilde{v}_t^{(1)}, \tilde{v}_t^{(2)})$ together with the information symbol u_t to encoders 1 and 2. Based on the inputs, the component encoders calculate the parity symbols. The code symbols $v_t^{(0)} = u_t$, $v_t^{(1)}$, and $v_t^{(2)}$ are then fed to the MMP input, and code block $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)}, v_t^{(2)})$ is sent over the channel.

Consider the ensemble of BCC's using an MMP of multiplicity 3, as shown in Figure 14. By definition, the *state μ_t of the MMP* at the t-th time instant is the number of 1's stored in its memory, and

$$\mu_t \in \{0, 1, \ldots, M\}. \tag{63}$$

We assume component encoder e has memory $m_{cc}^{(e)}$, $e \in \{1, 2\}$. Let $\sigma_t^{(e)}$ denote the state of component encoder e at the t-th time instant, where

$$\sigma_t^{(e)} \in \{0, 1, \ldots, 2^{m_{cc}^{(e)}} - 1\}. \tag{64}$$

The composite state of the two component encoders at time t is denoted $\boldsymbol{\sigma}_t = (\sigma_t^{(1)}, \sigma_t^{(2)})$. Combining the states of the MMP and the component encoders, the *state of the BCC encoder* is defined as $(\mu_t, \boldsymbol{\sigma}_t)$. As shown in Figure 15, a super trellis for the encoder ensemble can be constructed for analyzing the state transitions during encoding. The branches of the super trellis are labeled with $u_t/\tilde{\mathbf{v}}_t\mathbf{v}_t$. The output block $\mathbf{v}_t$ of the encoder at time t and the composite state of the two component encoders at time $t+1$ are functions of the composite state $\boldsymbol{\sigma}_t$, the input symbol u_t, and the output of the MMP $\tilde{\mathbf{v}}_t$:

$$\mathbf{v}_t = \mathrm{G}(\boldsymbol{\sigma}_t, u_t, \tilde{\mathbf{v}}_t), \tag{65}$$

$$\boldsymbol{\sigma}_{t+1} = \mathrm{F}(\boldsymbol{\sigma}_t, u_t, \tilde{\mathbf{v}}_t). \tag{66}$$

The functions of $\mathrm{G}(\cdot)$ and $\mathrm{F}(\cdot)$ depend on the component encoders. The code symbols $\mathbf{v}_t$ are then fed back to the MMP, and the next state of the MMP is given by

$$\mu_{t+1} = \mu_t + w_{_H}(\mathbf{v}_t) - w_{_H}(\tilde{\mathbf{v}}_t). \tag{67}$$

Conditioned on the permutor state μ_t, we can find the probability distribution of the permutor output $\tilde{\mathbf{v}}_t$. From a population of n symbols, the number of ordered samples of size i that can be formed without replacement is given by [22]

$$\begin{aligned}(n)_i &\triangleq n(n-1)\cdots(n-i+1) \\ &= n!/(n-i)!\,.\end{aligned} \tag{68}$$

Thus, the total number of ordered samples of the outputs from the multiplicity-3 MMP with overall constraint length M is

$\binom{M}{3}$. Among them, there are $\binom{\mu_t}{w_H(\tilde{\mathbf{v}}_t)}\binom{M-\mu_t}{3-w_H(\tilde{\mathbf{v}}_t)}$ ordered samples with the same weight (number of 1's) as $\tilde{\mathbf{v}}_t$. Under the assumption that the output symbols are randomly selected from the MMP, we have

$$P(\tilde{\mathbf{v}}_t|\mu_t) = \begin{cases} 0, \text{ if } 3-w_H(\tilde{\mathbf{v}}_t) > M-\mu_t \text{ or } w_H(\tilde{\mathbf{v}}_t) > \mu_t \\ \binom{\mu_t}{w_H(\tilde{\mathbf{v}}_t)}\binom{M-\mu_t}{3-w_H(\tilde{\mathbf{v}}_t)}\binom{M}{3}^{-1}, \text{ otherwise} \end{cases} \tag{69}$$

which follows from the fact that the number of 0's or 1's in $\mathbf{x}_t$ cannot exceed the number of 0's or 1's in storage. This probability distribution is used in the next section to recursively calculate the average distance spectrum of an ensemble of BCC's.

B. Calculation of the Average Distance Spectrum

In this section, we analyze the average distance spectrum of the codes in the ensemble of BCC's based on the Markov permutors described above. Since BCC's are linear, this spectrum coincides with the average weight spectrum of the codes in the ensemble. We assume that initially the BCC encoder is in the zero state, i.e., $\mu_0 = 0$, $\boldsymbol{\sigma}_0 = \mathbf{0}$. Assume an information symbol $u_0 = 1$ enters the encoder. Correspondingly, the MMP transitions to the state $\mu_1 = 1$ and the component encoders to a state $\boldsymbol{\sigma}_1 \neq \mathbf{0}$. The encoding process then continues from state $(\mu_1, \boldsymbol{\sigma}_1)$. Ultimately, with probability 1, the BCC encoder will return to the zero state $(\mu_l, \boldsymbol{\sigma}_l) = (0, \mathbf{0})$ at some l-th time instant. For the purpose of bounding the free distance, we are interested in the weight distribution of the encoder output sequence between the two time instants when the encoder is in the zero state.

Let $\bar{a}(d,i,l) = E[a(d,i,l)]$ denote the expectation of the number of paths with codeword weight d and information weight i that depart from the all-zero path at time instant 0 and remerge with the all-zero path at time l, $l \geq 1$. The set $\{\bar{a}(d,i,l)\}$, $0 \leq d,i \leq \infty$, $1 \leq l \leq \infty$, is called the *average extended weight spectrum* (AEWS) of the encoder. The AEWS is derived using a backward recursion on the super trellis. In the backward recursion, we must consider truncated paths that start from non-zero states, i.e., $(\mu_t, \boldsymbol{\sigma}_t) \neq (0, \mathbf{0})$, where the AEWS from state $(\mu_t, \boldsymbol{\sigma}_t)$ is denoted as $\bar{a}((\mu_t, \boldsymbol{\sigma}_t), d, i, l)$.

Now we describe the backward recursion. As shown in Figure 15, we assume that the encoder is in state $(\mu_t, \boldsymbol{\sigma}_t)$. With input $u_t = \{0,1\}$ and random outputs $\mathbf{x}_t$ from the MMP, several successive states $(\mu_{t+1}, \boldsymbol{\sigma}_{t+1})$ are possible in a one step transition. With u_t known, it follows directly from (66) that the transition probability is

$$P(\boldsymbol{\sigma}_t \rightarrow \boldsymbol{\sigma}_{t+1}|\mu_t) = P(\tilde{\mathbf{v}}_t|\mu_t), \tag{70}$$

where $P(\tilde{\mathbf{v}}_t|\mu_t)$ is given by (69). All paths starting from these successor states are extensions of the paths passing through state $(\mu_t, \boldsymbol{\sigma}_t)$. In summary, the AEWS's from the successor states $(\mu_{t+1}, \boldsymbol{\sigma}_{t+1})$ contribute to the AEWS from state $(\mu_t, \boldsymbol{\sigma}_t)$ in a probabilistic summation. It follows that

$$\begin{aligned} \bar{a}((\mu_t, \boldsymbol{\sigma}_t), d, i, l) = &\sum_{u_t=0}^{1} \sum_{\tilde{\mathbf{v}}_t} P(\tilde{\mathbf{v}}_t|\mu_t) \\ &\cdot \bar{a}((\mu_{t+1}, \boldsymbol{\sigma}_{t+1}), d - w(\mathbf{v}_t), i - u_t, l-1), \end{aligned} \tag{71}$$

where $\mathbf{v}_t$, $\boldsymbol{\sigma}_{t+1}$, and μ_{t+1} are given by (65), (66), and (67), respectively. Note that the codeword weights, information weights, and path lengths of the AEWS's from the successor states must be decreased to take into account the weights on the transition branch.

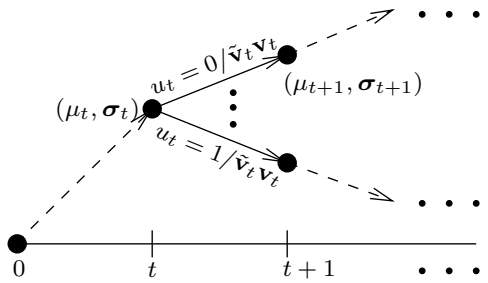


Fig. 15. State transitions on a super trellis.

In the super trellis, the path that diverges from the all-zero path is unique since it can be caused only by an information symbol $u_0 = 1$ entering the encoder. Thus the probability associated with this transition is unity. Let $(\mu_1, \boldsymbol{\sigma}_1)$ denote the corresponding successor state of the encoder, and d_1 denote the weight of the transition from $(0, \mathbf{0})$ to $(\mu_1, \boldsymbol{\sigma}_1)$. Substituting these values in (71), we obtain

$$\bar{a}(d,i,l) = \bar{a}((\mu_1, \boldsymbol{\sigma}_1), d - d_1, i-1, l-1) \tag{72}$$

On the basis of the AEWS, the *average weight spectrum* (AWS) is defined as

$$\bar{a}(d) = \sum_{l=1}^{\infty} \sum_{i=1}^{l} \bar{a}(d,i,l). \tag{73}$$

As in (73), if we sum over all i and l in (71), we obtain the following system of recursive equations for the AWS from state $(\mu_t, \boldsymbol{\sigma}_t)$:

$$\begin{aligned} \bar{a}((\mu_t, \boldsymbol{\sigma}_t), d) = &\sum_{u_t=0}^{1} \sum_{\tilde{\mathbf{v}}_t} P(\tilde{\mathbf{v}}_t|\mu_t) \\ &\cdot \bar{a}((\mu_{t+1}, \boldsymbol{\sigma}_{t+1}), d - w(\mathbf{v}_t)). \end{aligned} \tag{74}$$

Finally, the AWS can be computed using following steps:

1) Set the overall constraint length M and generate the super trellis $\{(\mu_t, \boldsymbol{\sigma}_t) \rightarrow (\mu_{t+1}, \boldsymbol{\sigma}_{t+1})\}$ according to (66) and (67) for the component encoders.
2) Find $(\mu_1, \boldsymbol{\sigma}_1)$ and d_1.
3) Set the boundary conditions

$$\bar{a}((0, \mathbf{0}), d) = \begin{cases} 1 & d = 0 \\ 0 & d \geq 1 \end{cases} \tag{75}$$

and

$$\bar{a}((\mu, \boldsymbol{\sigma}), d) = 0, \ \forall d < 0. \tag{76}$$

For $\mu = 0$ to M

4) For $d = 0$ to d_{max}
For $\mu = 0$ to M
For all $\boldsymbol{\sigma}$, calculate $\bar{a}((\mu, \boldsymbol{\sigma}), d)$ based on (74) and boundary conditions;

End
$\bar{a}(d) = \bar{a}((\mu_1, \boldsymbol{\sigma}_1), d - d_1)$;
End

C. A Lower Bound on Free Distance

After deriving the AWS for given component encoders with constraint length M, a free distance lower bound can be obtained using the usual Gilbert-Varshamov (see, e.g., [10]) argument, as stated in the following theorem.

Theorem 1: If $\hat{d}$ is the largest integer value of δ that satisfies

$$\sum_{d=1}^{\delta-1} \bar{a}(d) < 1, \tag{77}$$

then at least one code in the ensemble must have free distance not less than $\hat{d}$. ■

We calculate $\hat{d}$, and it follows from Theorem 1 that there exists at least one code in the ensemble for which d_{free} is lower bounded by $\hat{d}$. The free distance bound implied by (77) is a function of the component encoders and the overall constraint length M of the MMP. Recall that in Section III we showed that a BCC encoder with three convolutional permutors of width T has an overall constraint length of $M = 5(T-1)/2+1$. Solving for $\hat{d}$ for different values of M then gives us a numerical lower bound on d_{free}. We plot $\hat{d}$ as a function of M, $0 < M \leq 1000$, in Figure 16. Three rate $R = 1/3$ SBCC's with identical RSC component encoders of memory $m_{cc} = 2$, 3, and 4 were considered in the calculation. We see that the free distance bounds exhibit essentially linear growth as a function of the overall constraint length M of the MMP.

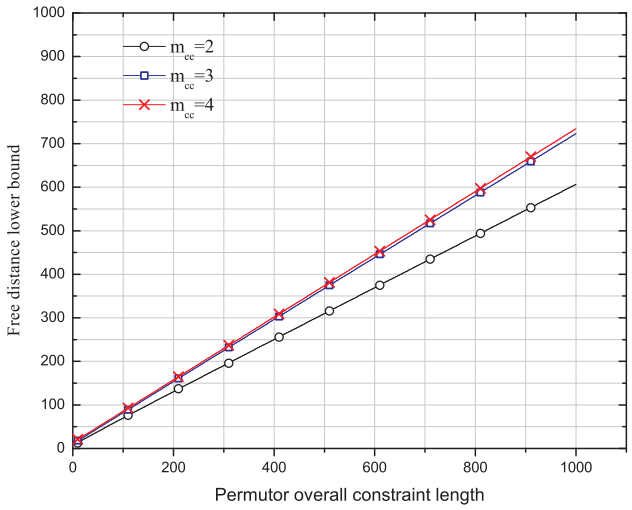


Fig. 16. Lower bounds on the free distance of BCC's with different component encoders.

Although the numerical results plotted in Figure 16 extend only to $M = 1000$, they provide strong evidence to conjecture that asymptotically, as M goes to infinity, the ratio of the free distance of these rate $R = 1/3$ BCC's to their overall constraint length is lower bounded by γ_{bc}, where γ_{bc} is the average slope of the corresponding curves in Figure 16. Values of γ_{bc} derived from Figure 16 are given in Table I for BCC's with rate $R_{cc} = 2/3$ component encoders of memory $m_{cc} = 2$, 3, and 4. The generator polynomials are denoted in octal form.

TABLE I
FREE DISTANCE BOUND FOR RATE $R = 1/3$ BCC'S WITH DIFFERENT COMPONENT ENCODERS.

Component encoder memory	Generator matrix	Asymptotic ratio γ_{bc}
$m_{cc} = 2$	$\begin{pmatrix} 1 & 0 & 4/7 \\ 0 & 1 & 5/7 \end{pmatrix}$	0.6069
$m_{cc} = 3$	$\begin{pmatrix} 1 & 0 & 17/15 \\ 0 & 1 & 13/15 \end{pmatrix}$	0.7230
$m_{cc} = 4$	$\begin{pmatrix} 1 & 0 & 25/35 \\ 0 & 1 & 23/35 \end{pmatrix}$	0.7341

It is interesting to compare this bound with the Costello bound [23] on the free distance of the ensemble of convolutional codes. The Costello bound states that there exists rate $R = b/c$ convolutional encoders of memory m with free distance lower bounded by the following inequality

$$\frac{d_{free}}{cm} \geq -\frac{R}{\log_2[2^{1-R}-1]} + O\left(\frac{\log_2 m}{m}\right), \tag{78}$$

which can also be written as

$$\frac{d_{free}}{bm} \geq -\frac{1}{\log_2[2^{1-R}-1]} + O\left(\frac{\log_2 m}{m}\right). \tag{79}$$

Since the overall constraint length M of a convolutional encoder is upper bounded by the inequality $M \leq bm$, we can write

$$\frac{d_{free}}{M} \geq -\frac{1}{\log_2[2^{1-R}-1]} + O\left(\frac{\log_2 M}{M}\right). \tag{80}$$

Asymptotically, as M goes to infinity, we have $d_{free} \geq \gamma_{cost} M$, where $\gamma_{cost} = -1/(\log_2[2^{1-R}-1])$. In particular, for $R = 1/3$, $\gamma_{cost} = 1.3028$. Note that the coefficients γ_{bc} for BCC's are roughly a factor of 2 less than the ratio γ_{cost} in the Costello bound. This is consistent with the typical reduction in distance growth rate observed when comparing Gallager's minimum distance bound [24] for LDPC block codes to the Gilbert-Varshamov [10] minimum distance bound for the ensemble of block codes.

IX. CONCLUSIONS

In this paper, we proposed a new class of turbo-like codes, namely, braided convolutional codes, that are suitable for high speed continuous data transmission. We presented a construction method for tightly and sparsely braided convolutional codes. For applications involving packetized data, we also introduced a blockwise encoding structure. Computer simulation results show that sparsely braided convolutional codes achieve good convergence performance with iterative decoding. Furthermore, the simulation results suggests that braided convolutional codes have good distance properties, in contrast to conventional turbo codes. This observation was theoretically confirmed by an analysis of braided convolutional

codes using a statistical Markov permutor model. For this model, we showed that braided convolutional codes have a free distance that grows linearly with overall constraint length, i.e., braided convolutional codes are asymptotically good.

ACKNOWLEDGMENT

Some of the results in Section VIII were obtained within the 2004 IMA Summer Program in Coding and Cryptology at the University of Notre Dame. We would like to thank Bodo Blume, Ali Pusane, and Zeying Wang for their contributions. Furthermore, we would like to thank Marcos Tavares for the computation of the density evolution threshold shown in Figure 13.

REFERENCES

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

[2] M. Lentmaier, D. V. Truhachev, and K. S. Zigangirov, "Iterative decodable sliding codes on graphs," in *Proceedings of ACCT-VIII*, St-Petersburg, Russia, Sept. 2002, pp. 190–193.

[3] D. V. Truhachev, M. Lentmaier, and K. S. Zigangirov, "On braided block codes," in *Proceedings of the IEEE International Symposium on Information Theory*, Yokohama, Japan, Jun. 2003, p. 32.

[4] P. Elias, "Error free coding," *IRE Transactions on Information Theory*, vol. 4, no. 4, pp. 29–37, Sept. 1954.

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

[6] G. Zemor, "On expander codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 835–837, Feb. 2001.

[7] W. Zhang, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Braided convolutional codes," in *Proceedings of the IEEE International Symposium on Information Theory*, Adelaide, Australia, Sept. 2005, pp. 592–596.

[8] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981.

[9] K. Engdahl, M. Lentmaier, and K. S. Zigangirov, "On the theory of low-density convolutional codes," in *Proceedings of the Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, Honolulu, Hawaii, June 1999, pp. 77–86.

[10] R. Johannesson and K. S. Zigangirov, *Fundamentals of convolutional coding*. New York, NY: IEEE Press, 1999.

[11] A. J. Feltström and K. S. Zigangirov, "Periodic time-varying convolutional codes with low-density parity-check matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 5, pp. 2181–2190, Sept. 1999.

[12] S. Lin and D. J. Costello, Jr., *Error control coding*, 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004.

[13] A. Pusane, A. J. F. Feltstrom, A. Sridharan, M. Lentmaier, K. S. Zigangirov, and D. Costello, Jr., "Implementation aspects of ldpc convolutional codes," *IEEE Trans. Commun.*, vol. 56, no. 7, pp. 1060–1069, July 2008.

[14] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[15] 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 Progress Report*, vol. 42, no. 127, pp. 1–20, Nov. 1996.

[16] A. J. Viterbi, "An intuitive justification of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.

[17] R. Horn and C. R. Johnson, *Topics in matrix analysis*. Cambridge, UK: Cambridge University Press, 1991.

[18] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

[19] M.B.S. Tavares, M. Lentmaier, G.P. Fettweis, and K.Sh. Zigangirov, "Asymptotic distance and convergence analysis of braided protograph convolutional codes," in *Proceedings of the 46th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sept. 2008.

[20] M.B.S. Tavares, M. Lentmaier, K.Sh. Zigangirov, and G.P. Fettweis, "LDPC convolutional codes based on braided convolutional codes," in *Proceedings of the IEEE International Symposium on Information Theory*, Toronto, Canada, July 2008.

[21] M. Lentmaier, A. Sridharan, K. S. Zigangirov, and D. J. Costello, Jr., "Terminated LDPC convolutional codes with thresholds close to capacity," in *Proceedings of the IEEE International Symposium on Information Theory*, Adelaide, Australia, Sept. 2005, pp. 1372–1376.

[22] H. Stark and J. W. Woods, *Probability, Random Processes, and Estimation Theory for Engineers*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1994.

[23] D. J. Costello, Jr., "Free distance bounds for convolutional codes," *IEEE Trans. Inform. Theory*, vol. 20, no. 3, pp. 356–365, May 1974.

[24] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

Wei Zhang was born on August 31, 1975 in Shijiazhuang, P.R.China. He received th B.E. degree in 1998 from Xidian University, Xi'an, P. R. China, M.S. degree in 1998 from Tsinghua University, Beijing, P. R. China, and Ph.D. degree in 2006 from University of Notre Dame, Notre Dame, IN. Since 2006 he is with the Corporate Research and Development Department, QUALCOMM Incorporated, working on 3GPP standards and advanced wireless receiver design. His research interests include digital transmission, channel coding, and wireless communications.

Michael Lentmaier received the Dipl.-Ing. degree in electrical engineering from University of Ulm, Ulm, Germany in 1998, and the Ph.D. degree in telecommunication theory from Lund University, Lund, Sweden in 2003. He then worked as a Post-Doctoral Research Associate at University of Notre Dame, Indiana, and at University of Ulm, Germany. From 2005 to 2007 he was with the Institute of Communications and Navigation of the German Aerospace Center (DLR) in Oberpfaffenhofen, working on high resolution channel estimation techniques for multipath mitigation in satellite navigation receivers. Since January 2008 he is a senior researcher and lecturer at the Vodafone Chair Mobile Communications Systems at Dresden University of Technology (TU Dresden). His research interests include design and analysis of coding systems, graph based iterative algorithms and Bayesian methods applied to decoding, detection and estimation.

Daniel J. Costello, Jr. was born in Seattle, WA, on August 9, 1942. He received the B.S.E.E. degree from Seattle University, Seattle, WA, in 1964, and 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 as an Assistant Professor of Electrical Engineering. He was promoted to Associate Professor in 1973, and to Full Professor in 1980. 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 1991, he was selected as one of 100 Seattle University alumni to receive the Centennial Alumni Award in recognition of alumni who have displayed outstanding service to others, exceptional leadership, or uncommon achievement. In 1999, he received a Humboldt Research Prize from the Alexander von Humboldt Foundation in Germany. In 2000, he was named the Leonard Bettex Professor of Electrical Engineering at Notre Dame.

Dr. Costello has been a member of IEEE since 1969 and was elected Fellow in 1985. Since 1983, he has been a member of the Information Theory Society Board of Governors, and in 1986 he served as President of the BOG. He has also served as Associate Editor for Communication Theory for the IEEE Transactions on Communications, Associate Editor for Coding Techniques for the IEEE Transactions on Information Theory, and Co-Chair of the IEEE International Symposia on Information Theory in Kobe, Japan (1988), Ulm, Germany (1997), and Chicago, IL (2004). In 2000, he was selected by the IEEE Information Theory Society as a recipient of a Third-Millennium Medal. He was co-recipient of the 2009 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.

Dr. Costello's research interests are in the area of 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.

Kamil Sh. Zigangirov was born in the U.S.S.R. in 1938. He received the M.S. degree in 1962 from the Moscow Institute for Physics and Technology,Moscow, U.S.S.R., and the Ph.D. degree in 1966 from the Institute of Radio Engineering and Electronics of the U.S.S.R. Academy of Sciences, Moscow, U.S.S.R.

From 1965 to 1991, he held various research positions at the Institute for Problems of Information Transmission of the U.S.S.R. Academy of Sciences, Moscow, first as a Junior Scientist, and later as a Main Scientist. During this period, he visited several universities in the United States, Sweden, Italy, and Switzerland as a Guest Researcher. He organized several symposia on information theory in the U.S.S.R. In 1994, he received the Chair of Telecommunication Theory at Lund University, Lund, Sweden. From 2003 to 2009, he has been a Visiting Professor at the University of Notre Dame, Notre Dame, IN, Dresden Technical University, Dresden, Germany, and at the University of Alberta, Edmonton, AB, Canada. His scientific interests include information theory, coding theory, detection theory, and mathematical statistics. In addition to papers in these areas, he published a book on sequential decoding of convolutional codes (in Russian) in 1974. With R. Johannesson, he coauthored the textbook *Fundamentals of Convolutional Coding* (Piscataway, NJ: IEEE Press, 1999). His book *Theory of CDMA Communication* was published by IEEE Press in 2004.