Turbo Decoding with Tail–Biting Trellises

Christian Weiß, Christian Bettstetter, Sven Riedel
Institute for Communications Engineering
Munich University of Technology
D–80290 München, Germany

Daniel J. Costello, Jr.
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556, USA

Abstract

Tail–biting codes are considered as component codes for parallel concatenated block codes. Based on the two–dimensional weight distribution of tail–biting codes, we calculate the minimum distance of the parallel concatenated code and give guidelines on how to choose good tail–biting component codes. We show how to encode tail–biting codes using systematic feedback encoders, which is an important design criterion. The performance of codes with different rate, length, and complexity using iterative (turbo) decoding is evaluated by simulation.

1 Introduction

Since the introduction of turbo decoding [1] in 1993, there has been considerable research on the investigation of parallel concatenated coding schemes employing convolutional codes (CCs) as component codes. Using block codes (BCs) as component codes has received less attention. Considering the rate loss imposed by terminating the CC, this is surprising particularly for short block lengths, and is mainly due to the favorable properties of feedback convolutional encoders with turbo decoding. The feedback enables us to pair information vectors of low Hamming weight with high parity weights. This is important for the performance of turbo codes at low to moderate signal–to–noise ratios (SNRs), while the minimum distance of the overall code governs the asymptotic behavior. Moreover, the regular trellis structure of CCs facilitates decoding, and there is a well developed theory of soft–in/soft–out (SISO) decoders for CCs. Recognizing that quasi–cyclic BCs are equivalent to tail–biting CCs [3], we can exploit these properties without suffering the rate loss of conventional CCs. In addition, employing block component codes allows a higher degree of parallel processing, which is desirable especially in delay sensitive applications.

In what follows, we describe a rate–k_0/n_0, $k_0 < n_0$, convolutional encoder as a device which generates the n_0–tuple $\boldsymbol{v}_t = (v_t^{(1)}, \ldots, v_t^{(n_0)})$ of code bits at time t given the k_0–tuple $\boldsymbol{u}_t = (u_t^{(1)}, \ldots, u_t^{(k_0)})$ of information bits, where $v_t^{(i)}, u_t^{(i)} \in \mathrm{GF}(2)$, $t \geq 0$. The mapping between the information sequence $\boldsymbol{u} = (\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots)$ and the code sequence $\boldsymbol{v} = (\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots)$ is determined by $\boldsymbol{v} = \boldsymbol{u}\mathbf{G}$, where $\mathbf{G}$ denotes the semi–infinite generator matrix. Moreover, sequences of $\boldsymbol{u}_t$ and $\boldsymbol{v}_t$ can be written as $\boldsymbol{u}(D) = \sum_{t=0}^{\infty} \boldsymbol{u}_t D^t$ and $\boldsymbol{v}(D) = \sum_{t=0}^{\infty} \boldsymbol{v}_t D^t$, respectively. Hence, an encoder can equivalently be described by a $(k_0 \times n_0)$ generator matrix $\mathbf{G}(D)$ of full rank with polynomial or rational entries such that $\boldsymbol{v}(D) = \boldsymbol{u}(D)\mathbf{G}(D)$. A generator matrix and its corresponding encoder is called systematic whenever all bits in $\boldsymbol{u}_t$ appear unchanged in $\boldsymbol{v}_t$. The state of the encoder at time t is denoted by $\boldsymbol{x}_t = (x_t^{(1)}, \ldots, x_t^{(m)})^T$, where m is the memory of the encoder.

2 The Tail–Biting Idea

From the strict definition of CCs it is clear that CCs can only be applied to semi–infinite sequences, i.e., encoding starts at time $t = 0$ in the all–zero state $\boldsymbol{x}_0 = (0, \ldots, 0)^T = \mathbf{0}^T$ and goes on continuously. In contrast to this, due to practical constraints (e.g., synchronization, channel estimation, etc.), almost any communication system is block–oriented, and, therefore, we must find methods to obtain code blocks of finite length from a convolutional encoder. If the information block length is Nk_0 bits, the most obvious possibility is to stop the encoding process after Nk_0 bits (direct truncation), but this leads to weaker error protection for the last codeword bits. The standard solution to avoid this performance degradation is to force the encoder back to the all zero–state by appending a block of mk_0 tail bits to the information vector (zero termination), which leads to an $((N+m)n_0, Nk_0)$ block code. Because of the rate loss imposed by termination, this method is inefficient, especially if the codewords are very short.

Tail–biting avoids the rate loss without suffering from degraded error protection at the end of the block. Using tail–biting, the state of the encoder at the beginning of the encoding process is not necessarily the all–zero state, but may be one of the other states. The fundamental idea behind tail–biting is that the encoder is controlled in such a way that it starts and ends the encoding process in the same state, i.e., $\boldsymbol{x}_0 = \boldsymbol{x}_N$. In the trellis representation of a tail–biting code only those paths that start and end in the same state are valid codewords (see, e.g., the bold line in Fig. 2b. In the following, we denote an (Nn_0, Nk_0) tail–biting code derived from a rate–k_0/n_0 convolutional code $\mathcal{C}$ over N cycles by $\mathcal{C}^N$.

3 Encoding Tail–Biting Codes Using Feedback Encoders

For encoders without feedback it is easy to fulfill the tail–biting boundary condition $\boldsymbol{x}_0 = \boldsymbol{x}_N$, since the ending state $\boldsymbol{x}_N$ depends only on the last m input k_0–tuples $\boldsymbol{u}_{N-m}, \ldots, \boldsymbol{u}_{N-1}$ to the encoder. For feedback encoders the situation is more complicated. In this case, the ending state $\boldsymbol{x}_N$ depends on the entire information vector $\boldsymbol{u} = (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1})$. Thus, we must calculate for a given information vector $\boldsymbol{u}$ the initial state $\boldsymbol{x}_0$ that will lead to the same state after N cycles.

We solve this problem by using the state space representation

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \mathbf{A}\boldsymbol{x}_t + \mathbf{B}\boldsymbol{u}_t^T & (1) \\ \boldsymbol{v}_t^T &= \mathbf{C}\boldsymbol{x}_t + \mathbf{D}\boldsymbol{u}_t^T & (2) \end{aligned}$$

of the encoder. The complete solution of (1) is calculated by the superposition of the zero–input solution $\boldsymbol{x}_t^{[zi]}$ and the zero–state solution $\boldsymbol{x}_t^{[zs]}$:

$$\boldsymbol{x}_t = \boldsymbol{x}_t^{[zi]} + \boldsymbol{x}_t^{[zs]} = \mathbf{A}^t \boldsymbol{x}_0 + \sum_{\tau=0}^{t-1} \mathbf{A}^{(t-1)-\tau} \mathbf{B} \boldsymbol{u}_\tau^T. \quad (3)$$

If we demand that the state at time $t = N$ is equal to the initial state $\boldsymbol{x}_0$, we obtain from (3) the equation

$$\left(\mathbf{A}^N + \mathbf{I}_m\right) \boldsymbol{x}_0 = \boldsymbol{x}_N^{[zs]}, \quad (4)$$

where $\mathbf{I}_m$ denotes the $(m \times m)$ identity matrix. Provided the matrix $(\mathbf{A}^N + \mathbf{I}_m)$ is invertible, the correct initial state $\boldsymbol{x}_0$ can be calculated knowing the zero–state response $\boldsymbol{x}_N^{[zs]}$.

Following this discussion, the encoding process is divided into two steps:

- The first step is to determine the zero–state response $\boldsymbol{x}_N^{[zs]}$ for a given information vector $\boldsymbol{u}$. The encoder starts in the all–zero state $\boldsymbol{x}_0 = \mathbf{0}$; all Nk_0 information bits are input, and the output bits are ignored. After N cycles the encoder is in the state $\boldsymbol{x}_N^{[zs]}$. We can calculate the corresponding initial state $\boldsymbol{x}_0$ using (4), and initialize the encoder accordingly.

- The second step is the actual encoding. The encoder starts in the correct initial state $\boldsymbol{x}_0$; the information vector $\boldsymbol{u}$ is input, and a valid codeword $\boldsymbol{v}$ results.

In an application, we could store the pre–computed solutions to (4) for various $\boldsymbol{x}_N^{[zs]}$ in a look–up table.

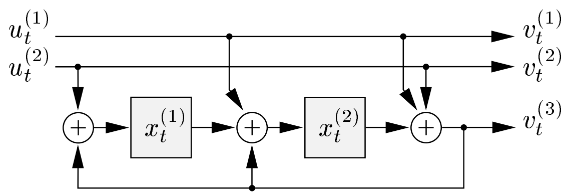


Figure 1: Encoder of the convolutional code $\mathcal{C}_A$

Example: The $(15, 10)$ tail–biting code $\mathcal{C}_A^5$ is obtained from a rate–$2/3$ convolutional code $\mathcal{C}_A$ over $N = 5$ cycles. A systematic generator matrix of $\mathcal{C}_A$ is given by

$$\mathbf{G}_A(D) = \begin{pmatrix} 1 & 0 & \frac{1+D}{1+D+D^2} \\ 0 & 1 & \frac{1+D^2}{1+D+D^2} \end{pmatrix}. \quad (5)$$

The corresponding encoder is shown in Fig. 1. By inspection we can write down the state matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad (6)$$

and using (4) we obtain

$$\left(\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^5 + \mathbf{I}_2\right) \boldsymbol{x}_0 = \boldsymbol{x}_5^{[zs]}, \quad (7)$$

whose solution for the initial state is

$$\boldsymbol{x}_0 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \boldsymbol{x}_5^{[zs]}. \quad (8)$$

The table of possible solutions is

$\boldsymbol{x}_5^{[zs]}$	$(0\ 0)^T$	$(0\ 1)^T$	$(1\ 0)^T$	$(1\ 1)^T$
$\boldsymbol{x}_0$	$(0\ 0)^T$	$(1\ 0)^T$	$(1\ 1)^T$	$(0\ 1)^T$

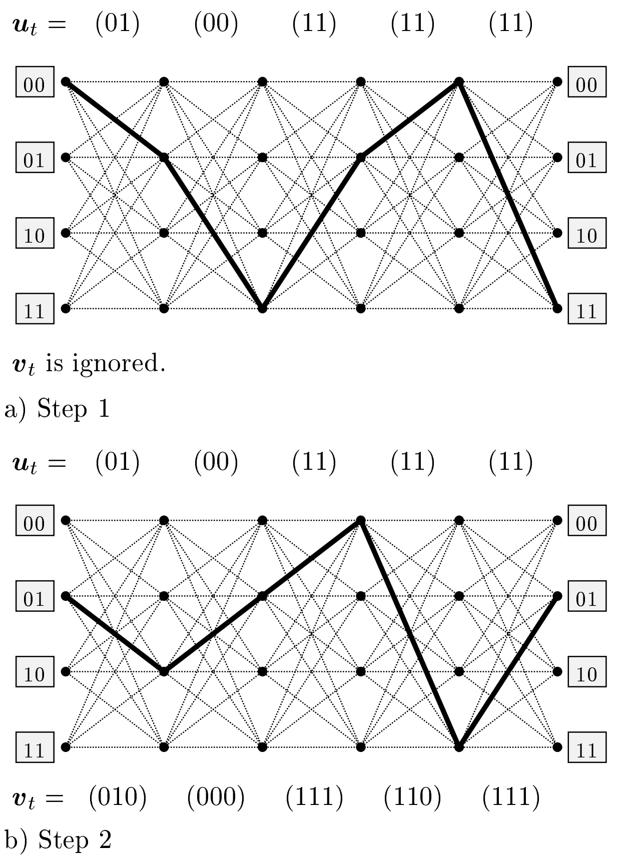
To demonstrate the encoding process, we encode the information sequence

$$\boldsymbol{u} = (\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_4) = (01, 00, 11, 11, 11). \quad (9)$$

This example is illustrated in Fig. 2. As mentioned above, we must perform two encoding runs. The first run starts in the all–zero state $\boldsymbol{x}_0 = (0\ 0)^T$. All $5k_0 = 10$ information bits are input, and the output bits are ignored. After $N = 5$ cycles, the encoder is in state $\boldsymbol{x}_5^{[zs]} = (1\ 1)^T$. We look up the corresponding initial state $\boldsymbol{x}_0$ in the above table, and then repeat the encoding starting in state $\boldsymbol{x}_0 = (0\ 1)^T$. This time, the boundary condition is satisfied, and a valid codeword

$$\boldsymbol{v} = (\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_4) = (010, 000, 111, 110, 111) \quad (10)$$

results.

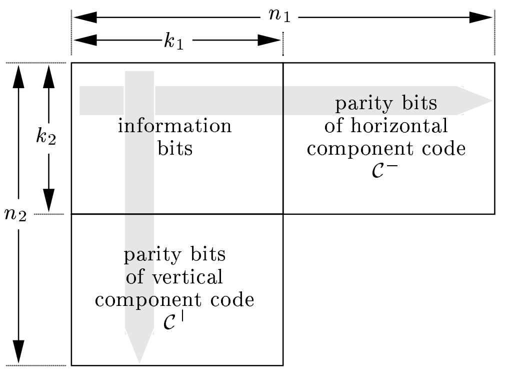


$u_t =$ (01) (00) (11) (11) (11)
00
01
10
11
v_t is ignored.
a) Step 1
$u_t =$ (01) (00) (11) (11) (11)
00
01
10
11
$v_t =$ (010) (000) (111) (110) (111)
b) Step 2

Figure 2: Two steps of the encoding procedure for the tail–biting code $\mathcal{C}_A^5$

4 Parallel Concatenation and Turbo–Decoding of Tail–Biting Codes

Fig. 3 shows the encoding scheme of a block interleaved parallel concatenated code using two systematic block codes, an (n_1, k_1) block code $\mathcal{C}^{-}$ and an (n_2, k_2) block code $\mathcal{C}^{|}$, as component codes. To generate a parallel concatenated block code (PCBC) the information bits are arranged in a $(k_2 \times k_1)$ matrix and then encoded in rows and columns applying the component codes $\mathcal{C}^{-}$ and $\mathcal{C}^{|}$, respectively. We obtain k_2 codewords of the horizontal code $\mathcal{C}^{-}$ and k_1 codewords of the vertical code $\mathcal{C}^{|}$. Using this encoding scheme an $(n_1k_2{+}k_1n_2{-}k_1k_2,\ k_2k_1)$ PCBC results, which can be decoded iteratively following the approach described in [2]. Since the component codewords are mutually independent, they can be decoded independently leading to a high degree of parallel processing. In principle any SISO decoder can be used to decode the component codes. The simulation results presented in Section 6 are achieved using a tail–biting MAP decoder based on the BCJR algorithm [4]. Binary (periodically time–varying) decoding trellises equivalent to the used CCs can be found in [5].



n_1
k_1
k_2
n_2
information bits
parity bits of horizontal component code $\mathcal{C}^{-}$
parity bits of vertical component code $\mathcal{C}^{|}$

Figure 3: Parallel concatenated block coding scheme

5 Distance Properties of Tail–Biting Codes for Use in Parallel Concatenation

5.1 The Two–Dimensional Weight Distribution

The weight distribution of a linear (n, k) block code $\mathcal{C}$ is a polynomial

$$E(W) = \sum_{w=0}^{n} a_w W^w = 1 + a_1 W + \ldots + a_n W^n \quad (11)$$

whose coefficients a_w denote the number of codewords with Hamming weight w. The weight w of a systematic codeword can be split up into two parts:

- The weight of the information bits $\boldsymbol{u}$, denoted by w_I, and
- the weight of the parity bits $\boldsymbol{p}$, denoted by w_P.

Clearly, $w = w_I + w_P$. Similar to (11), we can define the two–dimensional weight distribution

$$E(I, P) = \sum_{w_I} \sum_{w_P} a_{w_I, w_P} I^{w_I} P^{w_P}, \qquad (12)$$

which considers the information bits and the parity bits separately, and counts the number a_{w_I, w_P} of codewords in $\mathcal{C}$ with information weight w_I and parity weight w_P. The weight distribution $E(W)$ is a property of the code, whereas $E(I, P)$ depends on the mapping between information bits and code bits, and hence is a property of the encoder.

5.2 How to Choose Good Tail–Biting Component Codes for Parallel Concatenation

5.2.1 The Minimum Distance of PCBCs

The total Hamming weight w of a codeword in a PCBC consists of three parts:

- The weight of the k_1k_2 information bits, denoted by w_I,
- the weight of the $(n_1 - k_1)k_2$ parity bits from horizontal encoding ($\mathcal{C}^{-}$), denoted by w_P^{-}, and
- the weight of the $k_1(n_2 - k_2)$ parity bits from vertical encoding ($\mathcal{C}^{|}$), denoted by $w_P^{|}$.

Clearly, $w = w_I + w_P^{-} + w_P^{|}$.

Now let us calculate the resulting minimum distance d_{min} of a PCBC. By $d_{min}^{\mathcal{C}^-}$ and $d_{min}^{\mathcal{C}^{|}}$ we denote the minimum distance of the component codes $\mathcal{C}^-$ and $\mathcal{C}^{|}$, respectively. Our aim is to obtain a PCBC with high minimum distance. Hence, for low information weights w_I, we would like to avoid pairing low parity weights w_P^- with low parity weights $w_P^{|}$.

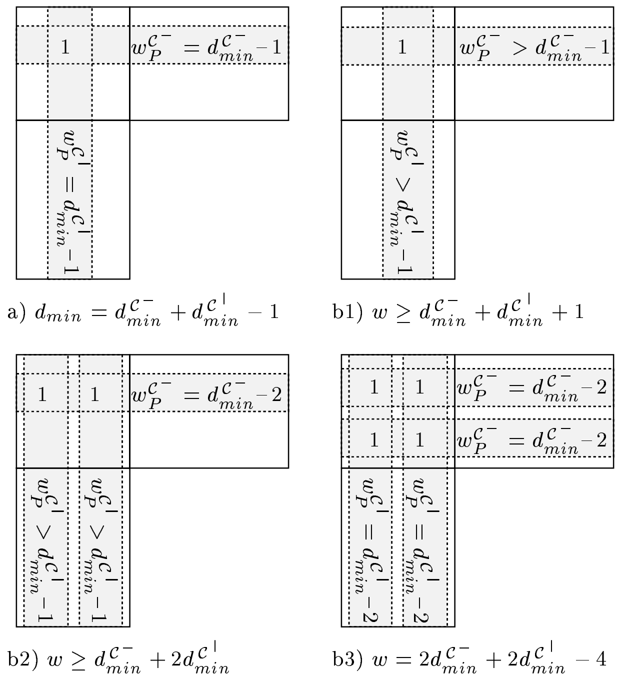
Assume $d_{min}^{\mathcal{C}^-}$ and $d_{min}^{\mathcal{C}^{|}}$ are caused by codewords with a single "1" among the k_1 (or k_2, respectively) information bits. In this case, a single "1" among the $k_1 k_2$ information bits of the PCBC results in minimum weight codewords of both the horizontal and the vertical code (see Fig. 4a). Since there is no other codeword of the PCBC with lower weight, the resulting minimum distance of the PCBC is

$$d_{min} = d_{min}^{\mathcal{C}^-} + d_{min}^{\mathcal{C}^{|}} - 1. \tag{13}$$

This situation is the worst case. The resulting d_{min} would be larger if the minimum distances of both component codes were not caused by weight–one information vectors. For example, suppose weight–two information vectors result in the minimum distance codewords and all weight–one information vectors generate codewords of higher weight. In this case, one of the bit patterns shown in Fig. 4b causes the minimum distance of the PCBC. Which bit pattern causes d_{min} depends on $d_{min}^{\mathcal{C}^-}$ and $d_{min}^{\mathcal{C}^{|}}$, but in all possible cases we have

$$d_{min} > d_{min}^{\mathcal{C}^-} + d_{min}^{\mathcal{C}^{|}} - 1. \tag{14}$$

Following this discussion, we observe that, in addition to the minimum distance of the component codes, the mapping of the component encoders



1
$w_P^{\mathcal{C}^-} = d_{min}^{\mathcal{C}^-} - 1$
$w_P^{\mathcal{C}^{|}} = d_{min}^{\mathcal{C}^{|}} - 1$
a) $d_{min} = d_{min}^{\mathcal{C}^-} + d_{min}^{\mathcal{C}^{|}} - 1$
1
$w_P^{\mathcal{C}^-} > d_{min}^{\mathcal{C}^-} - 1$
$w_P^{\mathcal{C}^{|}} > d_{min}^{\mathcal{C}^{|}} - 1$
b1) $w \geq d_{min}^{\mathcal{C}^-} + d_{min}^{\mathcal{C}^{|}} + 1$
1 1
$w_P^{\mathcal{C}^-} = d_{min}^{\mathcal{C}^-} - 2$
$w_P^{\mathcal{C}^{|}} > d_{min}^{\mathcal{C}^{|}} - 1$
$w_P^{\mathcal{C}^{|}} > d_{min}^{\mathcal{C}^{|}} - 1$
b2) $w \geq d_{min}^{\mathcal{C}^-} + 2d_{min}^{\mathcal{C}^{|}}$
1 1
$w_P^{\mathcal{C}^-} = d_{min}^{\mathcal{C}^-} - 2$
1 1
$w_P^{\mathcal{C}^-} = d_{min}^{\mathcal{C}^-} - 2$
$w_P^{\mathcal{C}^{|}} = d_{min}^{\mathcal{C}^{|}} - 2$
$w_P^{\mathcal{C}^{|}} = d_{min}^{\mathcal{C}^{|}} - 2$
b3) $w = 2d_{min}^{\mathcal{C}^-} + 2d_{min}^{\mathcal{C}^{|}} - 4$

Figure 4: Bit patterns that could result in the minimum distance of a PCBC

which produces their minimum weight codewords is a very important criterion in choosing good codes for parallel concatenation. Ideally, the encoders should be designed in such a way that they map low weight information words to high weight codewords. It is obvious that this requirement cannot be achieved with feedforward encoders of small memory, since in this case a single input "1" influences only a small number of codeword bits and therefore produces a codeword of low weight. This fact motivates the use of feedback encoders.

5.2.2 A Comparison between Feedforward and Feedback Encoders

To simplify the situation, in the remainder we suppose the two component codes are identical, that is, $\mathcal{C}^- = \mathcal{C}^{|}$. We compare three (512,256) PCBCs generated by using three different (24,16) tail–biting component codes with the encoding scheme depicted in Fig. 3 — the component code $\mathcal{C}_A^8$ generated by applying tail–biting to the feedback encoder of Fig. 1, and the codes $\mathcal{C}_B^8$ and $\mathcal{C}_C^8$, defined by the polynomial systematic generator matrices

$$\mathcal{C}_B^8 \quad : \quad \mathbf{G}_B(D) = \begin{pmatrix} 1 & 0 & 1+D+D^2 \\ 0 & 1 & 1+D \end{pmatrix}, \tag{15}$$

$$\mathcal{C}_C^8 \quad : \quad \mathbf{G}_C(D) = \begin{pmatrix} 1 & 0 & 1+D+D^2 \\ 0 & 1 & 1+D+D^3 \end{pmatrix}. \tag{16}$$

Table 1 shows parts of the two–dimensional weight distributions of $\mathcal{C}_A^8$, $\mathcal{C}_B^8$, and $\mathcal{C}_C^8$, respectively. Both memory–2 codes $\mathcal{C}_A^8$ and $\mathcal{C}_B^8$ achieve a minimum distance of 3. The feedforward encoder of $\mathcal{C}_B^8$ maps 8 information vectors of weight one and 16 information vectors of weight two to minimum weight codewords (entries $a_{1,2}$, $a_{2,1}$). In contrast, the feedback encoder of $\mathcal{C}_A^8$ generates all its minimum weight codewords with weight–three information vectors (entry $a_{3,0}$), whereas weight–one information vectors and

	$w_P \rightarrow$ $w_I \downarrow$	0	1	2	3	4	5
$\mathcal{C}_A^8$	0	1	0	0	0	0	0
	1	0	0	0	0	8	0
	2	0	0	24	0	72	0
	3	8	0	136	0	304	0
	4	10	0	392	0	980	0
$\mathcal{C}_B^8$	0	1	0	0	0	0	0
	1	0	0	8	8	0	0
	2	0	16	16	16	28	32
	3	0	24	56	128	152	96
	4	8	32	184	384	498	416
$\mathcal{C}_C^8$	0	1	0	0	0	0	0
	1	0	0	0	16	0	0
	2	0	0	32	0	56	0
	3	0	32	0	208	0	288
	4	4	0	384	0	1024	0

Table 1: Weight distributions of tail–biting codes

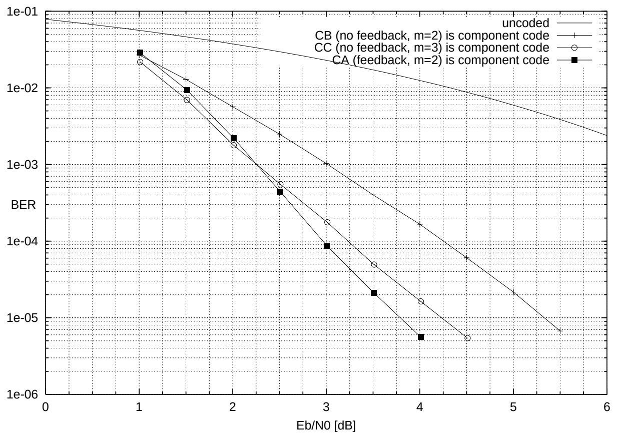


Figure 5: $(512, 256)$ PCBCs: Comp. codes are the (24,16) tail–biting codes $\mathcal{C}_A^8$, $\mathcal{C}_B^8$, and $\mathcal{C}_C^8$, respectively

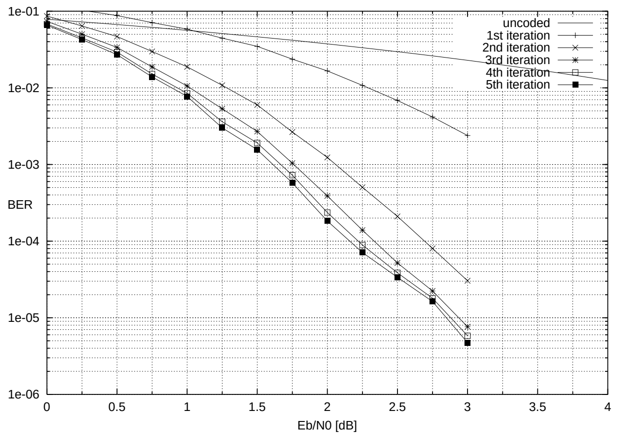


Figure 6: A $(507, 169)$ PCBC: Parallel concatenation of the (26,13) tail–biting code $\mathcal{C}_D^{13}$ of memory 2

weight–two information vectors are mapped to codewords of higher weight. The parallel concatenation of code $\mathcal{C}_B^8$ yields a minimum distance $d_{min} = 5$ (entry $a_{1,2}$), whereas the code $\mathcal{C}_A^8$ achieves $d_{min} = 9$ (entries $a_{1,4}$ and $a_{3,0}$). Thus, due to the larger minimum distance of the PCBC, and due to the advantageous mapping of information words to codewords, we expect code $\mathcal{C}_A^8$ to perform better than $\mathcal{C}_B^8$ in parallel concatenated schemes. The simulation results shown Fig. 5 confirm these observations. Although both component codes have a minimum distance of 3, and although they both have the same length and complexity, using component code $\mathcal{C}_A^8$ in a PCBC leads to a gain of 1.5 dB at a bit error rate (BER) of 10^{-5} compared to component code $\mathcal{C}_B^8$.

Surprisingly, component code $\mathcal{C}_A^8$ also outperforms component code $\mathcal{C}_C^8$, although the latter code has a larger minimum distance ($d_{min} = 4$) and twice as many states ($m = 3$). The reason is once again the mapping of information words to codewords. The systematic memory–3 feedforward encoder of $\mathcal{C}_C^8$ maps all weight–one input vectors to codewords of minimum weight, and, hence, a PCBC using component code $\mathcal{C}_C^8$ achieves a smaller minimum distance ($d_{min} = 7$) than a PCBC using component code $\mathcal{C}_A^8$. The advantage of $\mathcal{C}_A^8$ over $\mathcal{C}_C^8$ in a PCBC arises from the fact that the encoder of $\mathcal{C}_A^8$, by means of feedback, avoids pairing low weight information words with low weight codewords. For a BER of 10^{-5}, a PCBC generated with $\mathcal{C}_A^8$ outperforms a PCBC using component code $\mathcal{C}_C^8$ by 0.4 dB (see Fig. 5).

6 Simulation Results

Rate–1/3 PCBCs: We apply tail–biting to the simple rate–1/2 memory–2 encoder defined by the systematic generator matrix

$$\mathbf{G}_D(D) = \left(\begin{array}{cc} 1 & \frac{1+D^2}{1+D+D^2} \end{array} \right). \qquad (17)$$

The underlying convolutional code $\mathcal{C}_D$ has optimum free distance $d_{free} = 5$. Examination shows that for $N \geq 10$, tail–biting codes $\mathcal{C}_D^N$ achieve a minimum distance equal to the free distance of $\mathcal{C}_D$. We choose, for example, $N = 13$, and obtain a $(26, 13)$ tail–biting code $\mathcal{C}_D^{13}$. Its weight distribution is shown in Table 2. The parallel concatenation yields a (507,169) PCBC with $d_{min} = 17$. The simulated performance is shown in Fig. 6. A BER of 10^{-5} is achieved at an SNR of 2.9 dB after three iterations. Further iterations do not improve the result significantly.

Next, we concatenate $(24, 12)$ tail–biting codes $\mathcal{C}_E^{12}$ defined by

$$\mathbf{G}_E(D) = \left(\begin{array}{cc} 1 & \frac{1+D+D^2+D^4}{1+D+D^4} \end{array} \right) \qquad (18)$$

over $N = 12$ cycles which results in a $(432, 144)$ PCBC. From its performance curve (Fig. 7), we observe that almost no improvement compared with the (507,169) PCBC of Fig. 6 is achieved, although we increased the complexity of the tail–biting component code by a factor of 4. This result becomes understandable if we consider the weight distributions of $\mathcal{C}_D^{13}$ and $\mathcal{C}_E^{12}$ (Table 2). Both encoders map weight–one information vectors to parity weight eight and weight–two information vectors to parity weight four (at least). In order to improve the performance, we must increase the block length N. For example, consider the $(48, 24)$ code $\mathcal{C}_E^{24}$ defined by $\mathbf{G}_E(D)$ with $N = 24$. Now, all weight–one and all weight–two information vectors are mapped to high parity weights. In addition, the code now achieves $d_{min} = 7$ which is equal to the free distance of the underlying CC $\mathcal{C}_E$. The resulting $(1728, 576)$ PCBC has minimum distance $d_{min} = 25$. A BER of 10^{-5} is obtained at approximately 1.4 dB (six iterations), which is 0.6 dB better than the value of the cut–off rate R_0 for rate–1/3 codes. The coding gain of 1.3 dB compared to

BC	$E(I, P)$
$\mathcal{C}_D^{13}$	$1+13I^1P^8+13I^2P^4+13I^3P^2+13I^4P^2+\ldots$
$\mathcal{C}_E^{12}$	$1+12I^1P^8+18I^2P^4+84I^3P^4+12I^4P^2+\ldots$
$\mathcal{C}_E^{24}$	$1+24I^1P^{12}+96I^2P^{10}+48I^3P^4+72I^4P^4+\ldots$

Table 2: Weight distributions of tail–biting codes

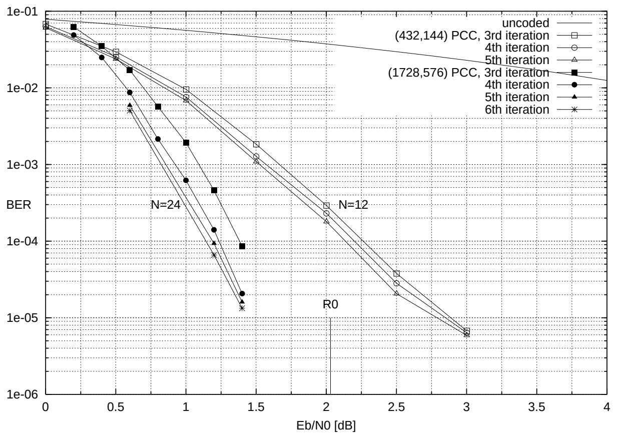


Figure 7: $(432, 144)$ and $(1728, 576)$ PCBCs using rate–1/2 memory–4 tail–biting component codes

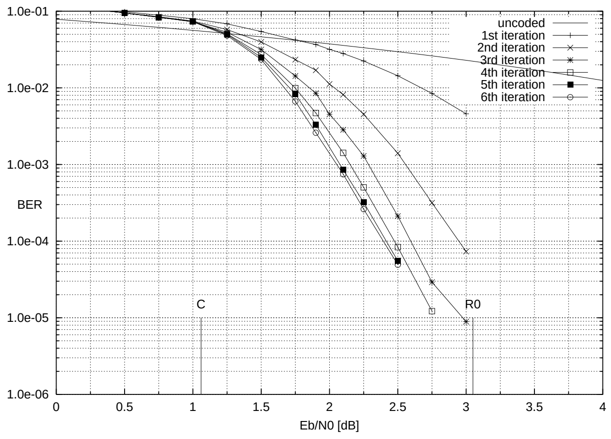


Figure 9: A $(1536, 1024)$ PCBC using rate–4/5 memory–4 tail–biting component codes

the $(432, 144)$ PCBC is certainly due to both the larger block length and the better properties of the component code.

We conclude that, although tail–biting is especially useful with short block lengths, the length of the code should be large enough

- to achieve a minimum distance equal to d_{free} of the underlying CC (if this is possible),
- to enable the encoder to map all weight–one information vectors to codewords of high weights,
- to enable the encoder to map low weight information vectors to non–minimum–weight codewords.

Rate–1/2 PCBCs: The performance of an $(800, 400)$ PCBC is shown in Fig. 8. As a component code, the $(30, 20)$ tail–biting code ($d_{min} = 4$, $m = 3$, $N = 10$) defined by

$$\mathbf{G}_F(D) = \begin{pmatrix} 1 & 0 & \frac{1+D^2}{1+D+D^3} \\ 0 & 1 & \frac{1+D^3}{1+D+D^3} \end{pmatrix} \qquad (19)$$

was employed.

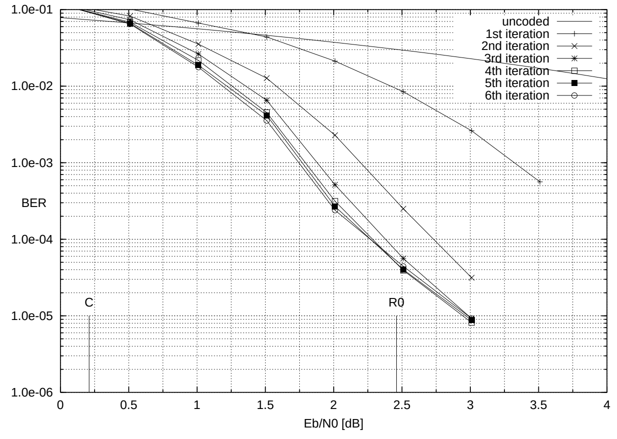


Figure 8: A $(800, 400)$ PCBC using rate–2/3 memory–3 tail–biting component codes

Rate–2/3 PCBCs: Finally, we apply tail–biting to the encoder

$$\mathbf{G}_G = \begin{pmatrix} 1 & 0 & 0 & 0 & \frac{1+D+D^2}{1+D+D^2+D^3+D^4} \\ 0 & 1 & 0 & 0 & \frac{1+D+D^3}{1+D+D^2+D^3+D^4} \\ 0 & 0 & 1 & 0 & \frac{1+D^2+D^3}{1+D+D^2+D^3+D^4} \\ 0 & 0 & 0 & 1 & \frac{1+D+D^2+D^3}{1+D+D^2+D^3+D^4} \end{pmatrix} \qquad (20)$$

to obtain a $(40, 32)$ code of rate $4/5$ ($N = 8$). The BER of its parallel concatenation is shown in Fig. 9. For comparison, Shannon's capacity limit, denoted as C, is given.

7 Conclusions

Choosing good tail–biting component codes for use in parallel concatenated coding schemes, it is important to use feedback encoders, which enable us to pair low–weight information vectors with high–weight parity vectors. We have shown how to encode tail–biting codes with systematic feedback encoders, and evaluated the performance of their parallel concatenation by simulation. Based on the two–dimensional weight distribution of tail–biting codes we have given guidelines how to choose good component codes for use in parallel concatenated schemes.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error–correcting coding and decoding: Turbo-Codes," *Proc. IEEE Int. Conf. Comm. (ICC)*, Geneva, Switzerland, pp. 1064–1070, May 1993.

[2] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. on Inf. Theory*, Vol. IT–42, No. 2, pp. 429–445, March 1996.

[3] G. Solomon and H. C. A. van Tilborg, "A connection between block and convolutional codes," *SIAM J. App. Math.*, Vol. 37, No. 2, pp. 358–369, October 1979.

[4] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Inf. Theory*, Vol. IT–20, pp. 284–287, March 1974.

[5] I. E. Bocharova and B. D. Kudryashov, "Rational rate punctured convolutional codes for soft–decision Viterbi decoding," *IEEE Trans. on Inf. Theory*, Vol. IT–43, pp. 1305–1313, July 1997.