# Untersuchung von Woven-Codes

Diplomarbeit
von
Jürgen Freudenberger


Abt. Informationstechnik

Universität Ulm
22. Januar 1999

# Contents


**1 Introduction** **1**

**2 Fundamentals** **3**

2.1 Channel Models and Channel Capacity . . . . . 3

2.1.1 Measurement of Information and Channel Capacity . . . . . 3

2.1.2 Capacity of the AWGN Channel . . . . . 5

2.2 Linear Binary Block Codes . . . . . 7

2.2.1 Distance Properties . . . . . 8

2.2.2 Bounds on Minimum Distance . . . . . 10

2.2.3 Union Bound . . . . . 11

2.2.4 Error Exponent . . . . . 12

2.3 Convolutional Codes . . . . . 13

2.3.1 Algebraic Description and Structural Properties . . . . . 13

2.3.2 Distance Properties . . . . . 17

2.3.3 Puncturing Convolutional Codes . . . . . 21

2.4 Soft-In/Soft-Out Decoding Schemes . . . . . 22

2.4.1 Viterbi Decoding . . . . . 23

2.4.2 Symbol-by-Symbol-MAP-Decoding . . . . . 24

2.4.3 Soft-In/Soft-Out Decoding . . . . . 25

**3 Parallel and Serial Concatenation** **27**

3.1 Basic Concepts of Parallel and Serial Concatenations . . . . . 28

3.1.1 Parallel and Serial Encoding Schemes . . . . . 28

3.1.2 Iterative Soft-In/Soft-Out Decoding . . . . . 29

3.2 Encoding and Decoding of Turbo Codes . . . . . 31
3.3 Serial Concatenated Convolutional Codes . . . . . 32
3.4 Theoretical Analysis . . . . . 33
3.4.1 Weight Distribution and Bounds . . . . . 33
3.4.2 Evaluating the WEF of Convolutional Codes . . . . . 36
3.4.3 Examples . . . . . 38

**4 Woven Codes** **41**
4.1 Encoding Scheme . . . . . 41
4.2 Decoding Scheme . . . . . 44
4.3 Distance Properties . . . . . 44
4.3.1 Free Distance and Active Distances . . . . . 44
4.3.2 Analogy to Serial Concatenated Codes . . . . . 47
4.4 Key Parameters . . . . . 48

**5 Simulation Results** **53**
5.1 Investigating Woven Codes . . . . . 53
5.1.1 Number of Outer Encoders . . . . . 54
5.1.2 Component Codes . . . . . 55
5.1.3 Interleaving . . . . . 57
5.2 Woven Codes in Comparison . . . . . 59
5.2.1 Memory of Component Codes . . . . . 60
5.2.2 Code Rate . . . . . 61
5.2.3 Code Dimension . . . . . 62
5.2.4 Iterative Decoding . . . . . 62

**6 Conclusions** **65**

**A Key Parameters of Woven Codes** **73**

**B Additional Simulation Results** **75**
B.1 Woven Codes with Interleaving . . . . . 75
B.2 Code Rate . . . . . 76
B.3 Optimum and Sub-optimum Decoding . . . . . 77

# Chapter 1

# Introduction

One of the most significant results from information theory is the coding theorem, revealing that there exist codes of sufficiently large block length $n$ capable of approaching channel capacity. In other words, if the data rate is less than channel capacity, communication over a noisy channel with an arbitrarily small error probability can be achieved by proper encoding and decoding.

The error-correcting capability of block codes improves as the block length increases, or in the case of convolutional codes with increasing constraint length $\nu$. However, the implementation complexity of maximum-likelihood (ML) decoders of such codes increases exponentially with $n$ respectively $\nu$ up to a point where decoding becomes physically unrealizable.

A practical method to obtain long codes with only a linear increase in decoding complexity is code concatenation. Forney invented an encoding scheme where the channel and the encoder/decoder of the inner code were considered as an extended super channel which is protected by an outer code. Many other concatenated code constructions like *generalized concatenated codes* and *turbo codes* have followed.

In the scope of this work, we will investigate a new class of code concatenation, the so called *woven convolutional codes with outer warp*, introduced by Höst, Johannesson, and Zyablov [HJZ97]. A number of outer convolutional codes and one inner convolutional code are woven together in a manner that resembles the structure of a fabric.

We start our discussion by giving a short introduction to linear binary block codes and convolutional codes in Chapter 2. Here we establish the notation and introduce some useful distance measures.

In the following two chapters we discuss the encoding and decoding of parallel and serial concatenated codes as well as that of woven codes, as all three encoder constructions are closely related. Moreover, we discuss distance properties of all three code classes.

The application of these ideas are presented in Chapter 5, where we give simulation results for woven convolutional codes and woven codes with interleaving.

We analyse the influence of key parameters and examine different types of interleaving. Furthermore, we present a detailed comparison of serial concatenated convolutional codes, turbo codes and woven codes with interleaving.

# Chapter 2

# Fundamentals

In this chapter we give a short introduction to linear binary block codes and convolutional codes. We establish the notation for the algebraic description as far as required in the following text. In particular some useful distance measures are introduced and maximum likelihood sequence estimation as well as symbol-by-symbol maximum a-posteriori decoding schemes are discussed. But before starting with coding theory, we first give a short introduction to channel models and channel capacity.

## 2.1 Channel Models and Channel Capacity

As the task of channel coding is adjusting the information stream to a given channel we have to discuss some channel models. The considered channels are mathematical models in contrast to physical models which describe the medium through which a signal passes or in which it is stored. We need to formalize the transmission model in order to compute how much information we could transmit over a given channel. Moreover, we need a quantitive measure of *information*.

### 2.1.1 Measurement of Information and Channel Capacity

**Average mutual information:** In information theory *information* is what we receive when uncertainty is reduced. Let $X$ and $Y$ denote two random variables, where $X$ is from an alphabet $\{x_1, \ldots, x_L\}$ of $L$ letters and $Y$ from an alphabet $\{y_1, \ldots, y_M\}$ of $M$ letters. We may interpret $X$ as the input letter into a discrete channel and $Y$ as the output. The occurrence of $Y = y_j$, changes the probability of $X = x_i$ from the a-priori probability $P(X = x_i)$ to the a-posteriori probability

$$P(X = x_i|Y = y_j) = \frac{P(Y = y_i|X = x_j)P(X = x_i)}{P(Y = y_i)} \ , \tag{2.1}$$

where the conditional probability $P(Y = y_i|X = x_j)$ is determined by the channel only. We define as a quantitive measure of the information provided by occurrence of the event $Y = y_j$ about the event $X = x_i$ as:

$$I(X = x_i; Y = y_j) = \log_2 \frac{P(X = x_i|Y = y_j)}{P(X = x_i)} = \log_2 \frac{f_{X|Y}(x_i|y_j)}{f_X(x_i)} \ , \tag{2.2}$$

where $f_X(x_i)$ denotes the probability function with:

$$\begin{aligned} f_X(x_i) &= P(X = x_i), \ \ i = 1, 2, \ldots, L \\ \sum_{i=1}^{L} f_X(x_i) &= 1 \end{aligned} \tag{2.3}$$

The self-information of the event $X = x_i$ is clearly a function of $X$ only and is denoted as

$$I(X = x_i) = -\log_2 f_X(x_i) \ . \tag{2.4}$$

The mean value $E[I(X = x)]$ of the self-information is called the entropy of $X$

$$H(X) = E[I(X = x)] = -\sum_{i=1}^{L} f_X(x_i) \log_2 f_X(x_i) \ . \tag{2.5}$$

The mean value $E[I(X = x; Y = y)]$ is called the average mutual information and can be expressed as the difference between the entropy of $Y$ and the conditional entropy of $Y$ given $X$

$$I(X; Y) = E[I(X = x; Y = y)] = H(Y) - H(Y|X) \ . \tag{2.6}$$

It is the average amount of information transmitted per symbol.

**Discrete memoryless channel:** The simplest class of channel models are the *discrete memoryless channels*. A discrete memoryless channel (DMC) is described by a set of conditional probabilities $f_{Y|X}(y_j|x_i)$, which are the probabilities that an input symbol $X = x_i$ from $\{x_1, \ldots, x_L\}$ appears as some output symbol $Y = y_j$ from $\{y_1, \ldots, y_M\}$ (see Figure 2.1), where the sizes of $L$ and $M$ of the alphabets need not be the same.

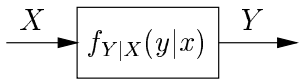



Figure 2.1: Discrete memoryless channel

A channel is *memoryless* if each output letter in a sequence depends only on the corresponding input, thus we obtain

$$f_Y(y_j) = \sum_{i=1}^{L} f_X(x_i) f_{Y|X}(y_j|x_i) \ . \tag{2.7}$$

**Example:** *A simple example for a discrete memoryless channel is the binary symmetric channel (BSC), as shown in Figure 2.2. Here the input and output alphabets are both binary and a single parameter, the crossover probability $P_\epsilon$, specifies the complete model.*

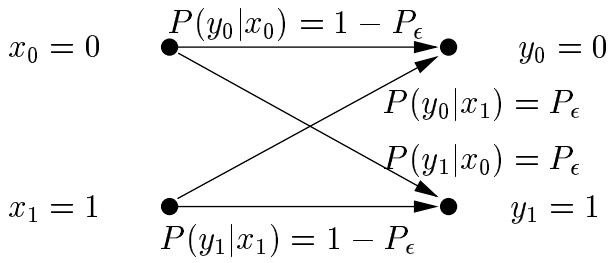



Figure 2.2: Binary symmetric channel

**Channel capacity and coding theorem:** We will now state an important measure, the so-called *channel capacity*, which is the largest average amount of information per symbol that can be transmitted over the channel. The mutual information depends on the characteristics of the channel and on the statistics of the input alphabet. If we maximize over the set of all possible input symbol probabilities $f_X(x)$ we obtain a measure which only depends on the channel and which we will call the *channel capacity*. Let $C$ denote the *channel capacity*, with

$$C = \max_{f_X(x)} I(X;Y) \ . \tag{2.8}$$

We are now able to discuss *Shannon's channel coding theorem*. We assume a binary information sequence which is fed into the channel encoder and will therefore be segmented into blocks of fixed length. Each information block consists of $k$ information digits, thus there is a total of $2^k$ distinct information blocks. The encoder maps each information word into a binary $n$-tuple, the code word with $n > k$. We call $R = k/n$ the code rate.

The channel coding theorem states that if the channel has capacity $C$ and if binary data enters the channel encoder at a rate $R < C$, then it is possible to reproduce the binary digits at the decoder output with a bit error probability as small as desired.

### 2.1.2 Capacity of the AWGN Channel

Next, we will discuss the *additive white Gaussian noise* (AWGN) channel. This memoryless channel is assumed to corrupt the input symbol by adding white Gaussian noise. Let $X$ denote a binary information symbol from $\{-1, +1\}$. If we consider for example binary phase-shift keying (BPSK) as modulation and a matched-filter

demodulator [Pro95] we may express the received symbol at the output of the correlator as

$$Y = X + N \ , \tag{2.9}$$

where we normalize the signal energy $E_s = 1$. The noise component $N$ is a zero-mean Gaussian random variable with variance $\sigma_n^2 = 1/2N_0$

$$f_N(n) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{n^2}{2\sigma_n^2}\right) \ . \tag{2.10}$$

Thus, $Y \in \{-\infty, \infty\}$ is a Gaussian random variable with mean either $X = -1$ or $X = +1$ depending on whether a binary one or a binary zero has been transmitted.

In order to evaluate the entropy of continuous random variable such as $Y$ and $N$ we require the definition of the *differential entropy*

$$H(X) = -\int_{-\infty}^{\infty} f_X(x) \log_2 f_X(x) dx \ . \tag{2.11}$$

The normal distribution maximizes the differential entropy over the set of all possible distributions with mean $\mu$ and variance $\sigma^2$ [Joh92] and we obtain

$$H(N) = \frac{1}{2} \log_2(2\pi e\sigma^2) \ . \tag{2.12}$$

As the channel output variable $Y$ is the sum of two independent random variables $X$ and $N$ we may express the mutual information as

$$I(X;Y) = H(Y) - H(Y|X) = H(Y) - H(N) \ , \tag{2.13}$$

which simplifies the evaluation of the channel capacity

$$\begin{aligned} C &= \max_{f_X(x)} I(X;Y) \\ &= \max_{f_X(x)} [H(Y) - H(N)] \\ &= \max_{f_X(x)} [H(Y)] - \frac{1}{2} \log_2(2\pi e\sigma_n^2) \ . \end{aligned} \tag{2.14}$$

$$\tag{2.15}$$

We notice that in order to maximize the mutual information we would need a normal distributed output variable $Y$ which requires a normal distributed input variable $X$. For applications we prefer discrete input alphabets such as the binary alphabet of BPSK.

To derive the channel capacity of the AWGN channel with a binary input alphabet we have to evaluate the following expression:

$$\begin{aligned} C = \max_{f_X(x)} \Bigg[ &\int_{-\infty}^{\infty} f_X(x=+1) f_{Y|X}(y|x=+1) \log_2 \frac{f_{Y|X}(y|x=+1)}{f_Y(y)} dy \\ &+ \int_{-\infty}^{\infty} f_X(x=-1) f_{Y|X}(y|x=-1) \log_2 \frac{f_{Y|X}(y|x=-1)}{f_Y(y)} dy \Bigg] \ . \end{aligned} \tag{2.16}$$

Due to the symmetry of $f_{Y|X}(y|x=-1)$ and $f_{Y|X}(y|x=+1)$ maximization requires $f_X(x=-1)=f_X(x=1)=1/2$ [Mas97] and the capacity of this channel is

$$\begin{aligned} C &= \int_{-\infty}^{\infty} \frac{1}{2} f_{Y|X}(y|x=+1) \log_2 \frac{f_{Y|X}(y|x=+1)}{f_Y(y)} dy \\ &\quad + \int_{-\infty}^{\infty} \frac{1}{2} f_{Y|X}(y|x=-1) \log_2 \frac{f_{Y|X}(y|x=-1)}{f_Y(y)} dy \ . \end{aligned} \tag{2.17}$$

Figure 2.3 illustrates the AWGN channel capacity C for a binary input alphabet as a function of the signal to noise ratio.

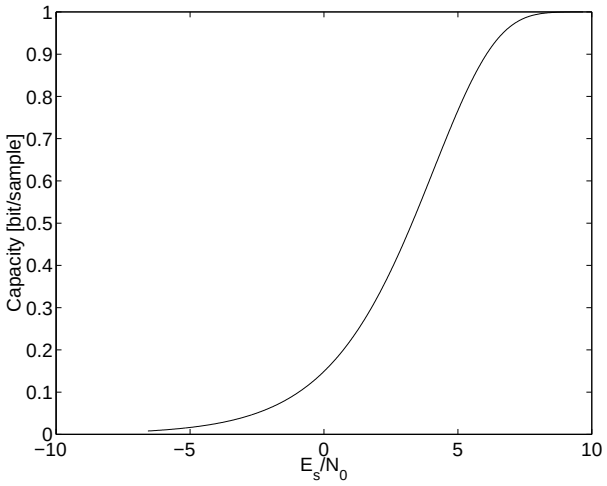



Figure 2.3: Channel capacity as a function of the signal to noise ratio

## 2.2 Linear Binary Block Codes

The channel coding theorem states that reliable information transmission is possible if $R < C$ holds assuming an appropriate design of the channel encoder and decoder. Appropriate encoder and decoder design is the matter of coding theory and will be the subject of this work. We start our discussion by presenting some important fundamentals of coding theory.

We assume that the output of an information source is a sequence of binary digits '0' or '1'. In block coding, this binary information sequence is segmented into blocks of fixed length. Each information block respectively information word consists of $k$ information digits and is denoted by $\mathbf{u}_t = (u_t^1, \ldots, u_t^k)$. There is a total of $2^k$ distinct information words. The encoder maps each information word $\mathbf{u}_t$ into a binary $n$-tuple, the code word $\mathbf{v}_t = (v_t^1, \ldots, v_t^n)$ with $n > k$. The set of all $2^k$ code words is called a block code, or more precisely:

Let $\mathbb{F}_2$ denote the finite binary field and $\mathbb{F}_2^n$ the $n$-dimensional vector space over the field $\mathbb{F}_2$. A block code of rate $R = k/n$ is called a linear binary block code $\mathcal{C}(n,k)$ if and only if its $2^k$ code words form a $k$-dimensional subspace of the vector space $\mathbb{F}_2^n$.

*Linearity* means that the sum of two code words is again a code word. The information sequence $\mathbf{u} = (\ldots, \mathbf{u}_1, \ldots, \mathbf{u}_t, \ldots)$ can be encoded into an output sequence $\mathbf{v} = (\ldots, \mathbf{v}_1, \ldots, \mathbf{v}_t, \ldots)$ by a generator $k \times n$ matrix $\mathbf{G}$ of rank $k$ with entries from $\mathbb{F}_2$, as follows:

$$\mathbf{v}_t = \mathbf{u}_t \mathbf{G} \quad . \tag{2.18}$$

For each block code exists a set of equivalent generator matrices, also called encoding matrices. Each of them encodes the same code but with different mappings from information to code words. An encoding matrix is called a *systematic encoding matrix* if the information word with unaltered information bits is part of the resulting code word. Let $\mathbf{I}_k$ denote the $k \times k$ identity matrix and $\mathbf{A}$ any $k \times (n-k)$ matrix, then a generator matrix $\mathbf{G}$ is systematic if it can be written as :

$$\mathbf{G} = [\mathbf{I}_k \;\; \mathbf{A}] \, . \tag{2.19}$$

### 2.2.1 Distance Properties

The distance properties of a code determine its error-correcting and error-detecting capabilities. The *Hamming weight* of a code word $\mathbf{v}$, denoted $wt(\mathbf{v})$, is defined as the number of nonzero components of $\mathbf{v}$. The *Hamming distance* between two code words $\mathbf{v}$ and $\mathbf{v}'$, denoted $dist(\mathbf{v}, \mathbf{v}')$, is defined as the number of components where they differ. From linearity follows:

$$dist(\mathbf{v}, \mathbf{v}') = wt(\mathbf{v} - \mathbf{v}') \quad . \tag{2.20}$$

The *minimum distance* of a code is defined as

$$d_{min} = \min_{\mathbf{v},\mathbf{v}' \in \mathcal{C},\ \mathbf{v} \neq \mathbf{v}'} dist(\mathbf{v}, \mathbf{v}') = \min_{\mathbf{v} \in \mathcal{C},\ \mathbf{v} \neq 0} wt(\mathbf{v}) \quad . \tag{2.21}$$

**Example:** *As a first example we present a simple $\mathcal{C}(3,1)$ repetition code, with two code words: $\mathbf{v} = (0,0,0)$ and $\mathbf{v}' = (1,1,1)$, corresponding to information words $\mathbf{u} = 0$ and $\mathbf{u}' = 1$, respectively. This code has rate $R = 1/3$ and minimum distance $d_{min} = wt(\mathbf{v}') = 3$. Considering a binary symmetric channel we assume that a randomly selected information word $\mathbf{u}$ is encoded into a code word $\mathbf{v}$ and that the word $\mathbf{r}$ is received after transmission over the channel. For this $\mathcal{C}(3,1)$ code we obtain $2^n = 8$ possible channel output sequences, denoted $\mathbf{r}$, and the following decoding rule for an estimated code word $\hat{\mathbf{v}}$:*

$$\mathbf{r} = \begin{cases} 000 \\ 001 \\ 010 \\ 100 \end{cases} \rightarrow \hat{\mathbf{v}} = \mathbf{v} \qquad \mathbf{r} = \begin{cases} 111 \\ 110 \\ 101 \\ 011 \end{cases} \rightarrow \hat{\mathbf{v}} = \mathbf{v}'$$

*This decoding rule maximizes the probability $P(\mathbf{r}|\hat{\mathbf{v}})$ and therefore employs a maximum likelihood decision rule. Given a received sequence, the most likely transmitted code word is computed. The code is capable of correcting all error patterns with error weight $e = (d_{min} - 1)/2$ and it is able to detect all error patterns of $d_{min} - 1$ or fewer errors.*

In general, a block code with minimum distance $d_{min}$ is capable of correcting all error patterns with weight

$$e \leq \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \quad . \tag{2.22}$$

We will now define a family of weight enumerating functions which are important for the following discussion. An overview of these functions is presented in Table 2.1.

| symbol | function | abbreviation |
|---|---|---|
| $A(W)$ | weight enumerating function | WEF |
| $A(I, W)$ | input-output weight enumerating function | IOWEF |
| $A(i, W)$ | conditional weight enumerating function | CWEF |
| $A(I, J)$ | input-redundancy weight enumerating function | IRWEF |

Table 2.1: Overview of weight enumerating functions

- Let $\mathcal{C}$ be an (n,k) linear code, its *weight enumerating function* (WEF) $A(W)$ is defined as

$$A(W) = \sum_{w=0}^{n} A_w W^w \quad , \tag{2.23}$$

  where $A_w$ is the number of code words of weight $w$ in $\mathcal{C}$. The numbers $A_0, \ldots, A_n$ are called the *weight distribution* of $\mathcal{C}$.

- Let $A(I, W)$ denote the *input-output weight enumerating function*

$$A(I, W) = \sum_{i=0}^{k} \sum_{w=0}^{n} A_{i,w} I^i W^w = \sum_{i=0}^{k} I^i A(i, W) \quad , \tag{2.24}$$

  where $A_{i,w}$ represents the number of codewords with weight $w$ generated by information words of weight $i$. $A(i, W)$ denotes its conditional weight enumerating function.

- Let $A(I, J)$ denote the *input-redundancy weight enumerating function*

$$A(I, J) = \sum_{i=0}^{k} \sum_{j=0}^{n-k} A_{i,j} I^i J^j = \sum_{i=0}^{k} I^i A(i, J) \quad , \tag{2.25}$$

  where $A_{i,j}$ represents the number of codewords with parity weight $j$ generated by information words of weight $i$. $A(i, J)$ denotes its conditional weight enumerating function.

From linearity follows that the weight distribution and the distance distribution are equal. This equality makes the study of distance structures of linear codes much easier than that of non-linear codes.

The WEF is a code property and therefore independent of the mapping of information bits to code bits, it can be used to upper bound the word error probability in case of maximum likelihood decoding. On the other hand the bit error rate will be influenced by the mapping of information words to code words. Therefore, we have defined the *input-output weight enumerating function* (IOWEF) which results from the particular encoding of a block code. In addition we have defined the *input-redundancy weight enumerating function* (IRWEF) for systematic encoded block codes and the conditional weight enumerating functions (CWEF) for both weight distributions.

**Example:** *As a second example for a linear binary block code we present the* $(7, 4)$ *Hamming code, encoded by the following systematic generator matrix:*

$$\mathbf{G} = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right)$$

*This rate* $R = 4/7$ *Hamming code has minimum distance* $d_{min} = 3$ *and the weight distributions presented in the following table:*

| | |
|---|---|
| WEF | $A(W) = 1 + 7W^3 + 7W^4 + W^7$ |
| IOWEF | $A(I, W) = 1 + I(3W^3 + W^4) + I^2(3W^3 + 3W^4) + I^3(W^3 + 3W^4) + I^4W^7$ |
| IRWEF | $A(I, J) = 1 + I(3J^2 + J^3) + I^2(3J + 3J^2) + I^3(1 + 3J) + I^4J^3$ |

Table 2.2: WEF family of the systematic encoded rate $R = 4/7$ Hamming code

### 2.2.2 Bounds on Minimum Distance

In coding theory we often use bounds to indicate what distance properties or code performance we may achieve. In principle, we can distinguish between two kinds of bounds - bounds on minimum distance and bounds on performance. For example we can upper bound the minimum distance with the *Hamming bound*

$$\sum_{i=0}^{e} \binom{n}{i} \leq 2^{n-k} \quad , \qquad e \leq \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \quad . \tag{2.26}$$

A lower bound on the minimum distance is the *Varshamov bound* which states that there exists an $(n, k)$ linear block code with minimum distance $d_{min}$ which satisfies:

$$\sum_{i=0}^{d_{min}-2} \binom{n-1}{i} \geq 2^{n-k-1} \quad . \tag{2.27}$$

### 2.2.3 Union Bound

We use two different measures of code performance. The word error probability $P_W$ is the probability of the transmitted code word being decoded to some other code word at the receiver. The bit error probability $P_B$ is the probability of a transmitted information bit being erroneously estimated. The word error probability only depends on the distance properties of the code while the bit error probability also depends on the particular encoding.

A block code with minimum distance $d_{min}$ is usually capable of correcting many error patterns of $e+1$ or more errors. If we want to determine the word error probability as a measure of performance we have to take into account the complete distance distribution. For codes with large dimension the calculation of the weight distribution often becomes unrealizable.

An upper bound on performance which considers the complete weight distribution of a code is the *union bound*. It is based on the following idea: If an event can be expressed as the union of several sub-events, then the probability of that event is always less than or equal to the sum of the probabilities of all sub-events.

We consider maximum likelihood decoding at the receiver. Due to linearity of block codes, we can derive error probabilities by assuming the all zero code word to be transmitted. Thus, the word error probability $P_W$ is the probability that the all zero code word is decoded to some other code word. For each code word, other than the all zero one, we define the event $\mathcal{E}_i$ as a set of error patterns which will cause an error by maximum likelihood decoding. We obtain

$$P_W = P(\cup_i \mathcal{E}_i) \ . \tag{2.28}$$

$P(\mathcal{E}_i)$ is the same for all code words of weight $w$

$$P(w) = P(\mathcal{E}_i) \ , \forall \ wt(\mathbf{v}) = w, \ \ \mathbf{v} \in \mathcal{C} \ . \tag{2.29}$$

For transmission with BPSK modulation over an un-quantized channel with additive white Gaussian noise this probability is:

$$P(w) = \frac{1}{2} erfc\left(\sqrt{w\frac{E_s}{N_0}}\right) \ . \tag{2.30}$$

Now we can upper bound $P_W = P(\cup_i \mathcal{E}_i)$ by the sum over all $P(\mathcal{E}_i)$

$$\begin{aligned} P_W &\leq \sum_i P(\mathcal{E}_i) \\ &\leq \sum_{w=d_{min}}^{n} A_w \cdot P(w) \ , \end{aligned} \tag{2.31}$$

where $A_w$ denotes the number of code words of weight $w$.

Using the IOWEF we are able to extent the union bound on word error probability from Equation 2.31 to upper bound the bit error probability $P_B$ for the AWGN channel:

$$P_B \leq \frac{1}{k} \sum_{w=d_{min}}^{n} \sum_{i=1}^{k} i \;\; A_{i,w} \cdot P(w) \;\; . \tag{2.32}$$

With the definition of the *bit error multiplicities*

$$B_w = \frac{1}{k} \sum_{i=1}^{k} i \;\; A_{i,w} \;\; , \tag{2.33}$$

we obtain

$$P_B \leq \sum_{w=d_{min}}^{n} B_w \cdot P(w) \;\; . \tag{2.34}$$

**Example:** *In order to illuminate the constituents of above formulas we consider a block code with binomial weight distribution which we depict in Figure 2.4. Moreover, this figure illustrates the error weight probability $P(w)$ for an AWGN channel and the product terms $A_w \cdot P(w)$.*

*We observe that some product terms $A_w \cdot P(w)$ dominate the sum of the union bound. With increasing signal to noise ratio the lower weight terms get more important and asymptotically the minimum distance determines the bit error rate. In the 'classical' coding theory we look for codes with high minimum distance in order to improve the code performance. In this sense codes which fulfill the Varshamov bound are considered to be 'good'. On the other hand for low signal to noise ratios other regions of the weight distribution determine the word error probability. The bit error rate is determined by the so-called bit error multiplicities $B_w$.*

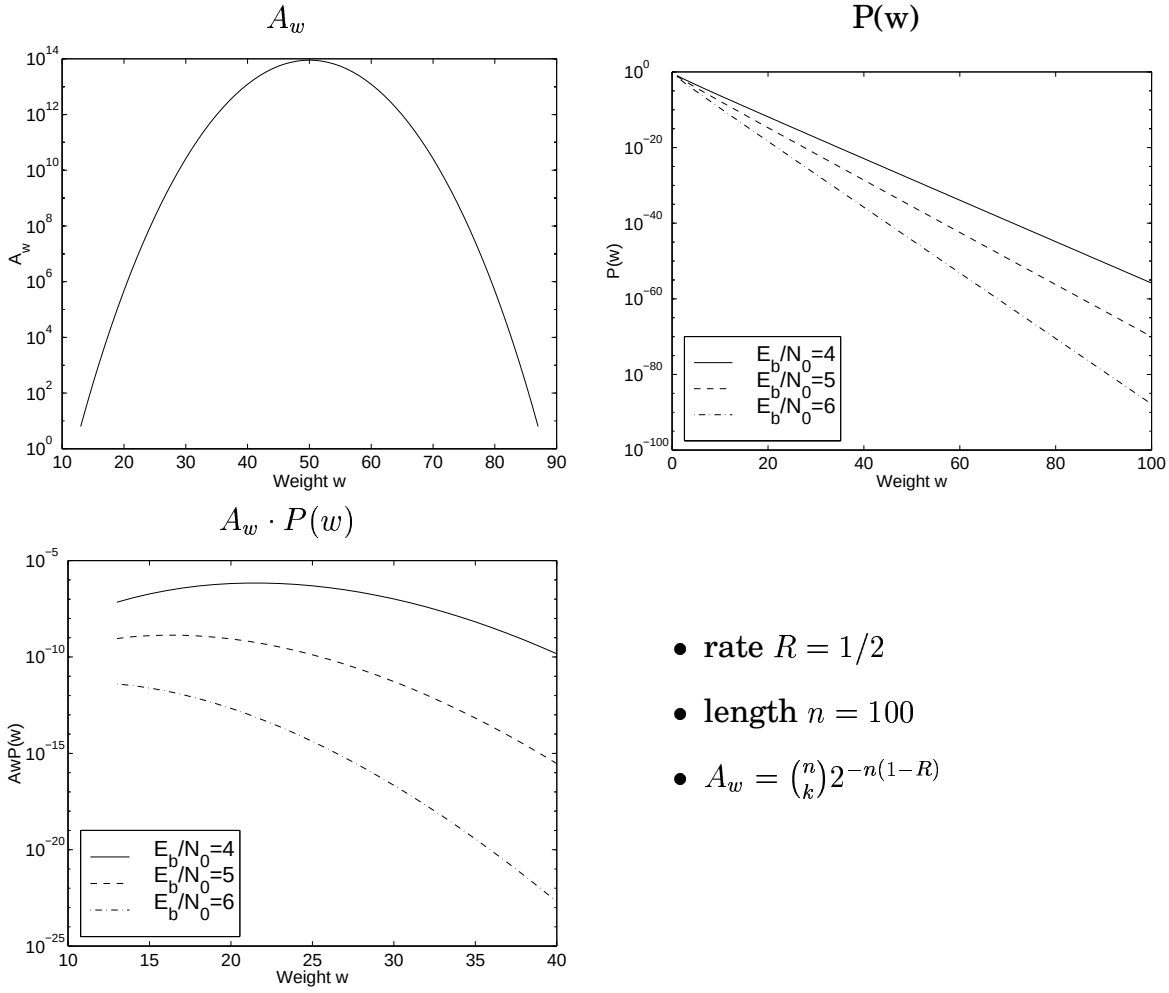
### 2.2.4 Error Exponent

The channel coding theorem states that if $R < C$ is fulfilled, then it is possible to reproduce the transmitted information sequence at the decoder output with a bit error probability as small as desired. In particular it states that for $n \to \infty$ the bit error rate converges to zero. For practical reasons we are interested in the question, what bit error probability is achievable for a given code rate $R$ and length $n$.

Gallager derived an upper bound on $P_B$ which is a function of the code rate and length [Gal68]

$$P_B \leq E[P_B] = e^{-nE(R)} \;\; , \tag{2.35}$$

where $E[P_B]$ is the averaged error probability of a randomly generated code set and $E(R)$ is the so-called error exponent. Gallager's bound states that the bit error rate of a randomly chosen code decreases exponentially with increasing code length, where the slope of the decrement depends on the code rate.




- rate $R = 1/2$
- length $n = 100$
- $A_w = \binom{n}{k} 2^{-n(1-R)}$

Figure 2.4: Error weight probability $P(w)$ and the products $A_w \cdot P(w)$ dominating the word error probability

# 2.3 Convolutional Codes

## 2.3.1 Algebraic Description and Structural Properties

In the remaining parts of this chapter we will deal with convolutional codes. In this section we will establish the notation for the algebraic description and we will discuss some structural properties of this code class.

In contrast to block codes is the $t$th code block $\mathbf{v}_t$ of a convolutional code sequence $\mathbf{v}$ a linear function of the corresponding $\mathbf{u}_i, i \in \{t - m, \ldots, t\}$, information words and not only of $\mathbf{u}_t$, where the integer $m$ denotes the *memory* of the encoder.

The encoder of a rate $R = k/n$ convolutional code is depicted in Figure 2.5. Each input corresponds to a shift register. A particular encoder can be implemented in different forms. We will only consider the *controller canonical form* of an encoder

realization. The controller canonical form of a rational transfer function is given in Figure 2.6.

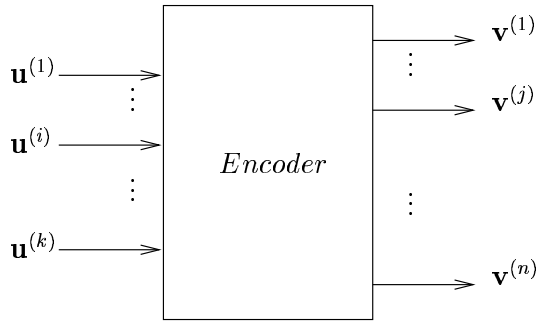



Figure 2.5: Convolutional encoder with $k$ inputs and $n$ outputs

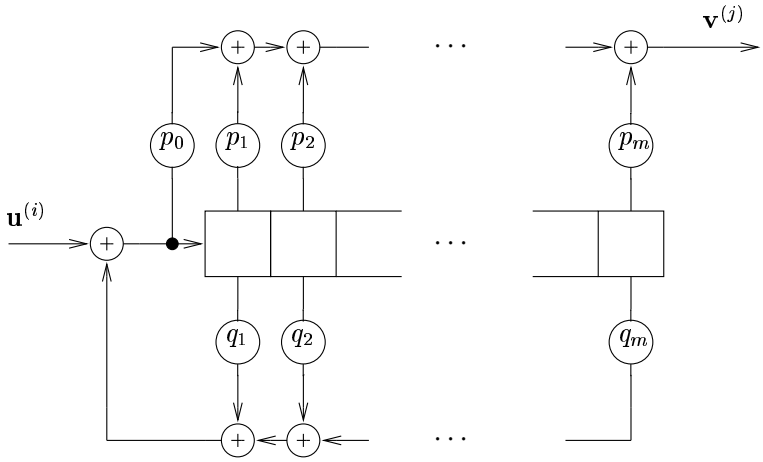



Figure 2.6: Rational transfer function in controller canonical form

Consider the $i$th input sequence of a convolutional encoder. This sequence is shifted into a register of length $\nu_i$. Thus, the corresponding generator impulse responses $\mathbf{g}_i^{(j)}$ are limited to length $\nu_i + 1$ and we call $\nu_i$ the *constraint length* of the $i$th input sequence. As the numbers of memory elements of the $k$ registers may differ we define the *memory* $m$ of the encoder:

$$m = \max_i \nu_i \quad , \tag{2.36}$$

the *minimum constraint length* $\nu_{min}$

$$\nu_{min} = \min_i \nu_i \quad , \tag{2.37}$$

and the *overall constraint length* $\nu$:

$$\nu = \sum_{i=1}^{k} \nu_i \quad . \tag{2.38}$$

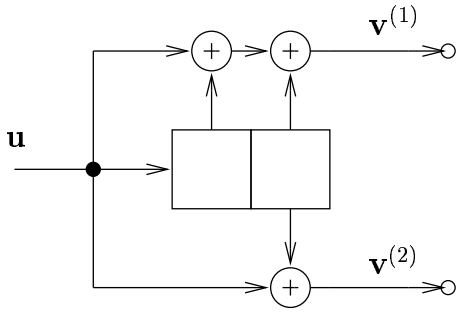



Figure 2.7: A rate $R = 1/2$ convolutional encoder

**Example:** *Consider the convolutional encoder in Figure 2.7 which encodes a rate $R = 1/2$ convolutional code. Information blocks of length $k = 1$ are shifted into a register of length $m = 2$. The output sequence is a serialization of the two sequences $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$. Each output block of length $n = 2$ is generated by modulo two addition of the current input block and some symbols of the register contents:*

$$\begin{aligned} \mathbf{v}_t^{(1)} &= \mathbf{u}_t + \mathbf{u}_{t-1} + \mathbf{u}_{t-2} \\ \mathbf{v}_t^{(2)} &= \mathbf{u}_t + \mathbf{u}_{t-2} \ . \end{aligned}$$

*This equations can be regarded as convolutions of the input sequence with the generator impulse responses $\mathbf{g}^{(1)} = (1, 1, 1, 0, \ldots)$ and $\mathbf{g}^{(2)} = (1, 0, 1, 0, \ldots)$, respectively.*

Consider the $D$-transform

$$\mathbf{x}^{(i)} \quad \circ\!\!-\!\!\bullet \quad \mathbf{X}_i(D) = \sum_{t=j}^{+\infty} x_t^{(i)} D^t \ , \quad j \in \mathbb{Z} \ . \tag{2.39}$$

It is often convenient to express the sequences of information and code blocks in terms of the delay operator $D$:

$$\begin{aligned} \mathbf{U}(D) &= \mathbf{u}_0 + \mathbf{u}_1 D + \mathbf{u}_2 D^2 + \ldots & (2.40) \\ \mathbf{V}(D) &= \mathbf{v}_0 + \mathbf{v}_1 D + \mathbf{v}_2 D^2 + \ldots & (2.41) \\ \mathbf{V}(D) &= \mathbf{U}(D)\mathbf{G}(D) \ , & (2.42) \end{aligned}$$

where $\mathbf{G}(D)$ is the encoding (or generator) matrix of the convolutional code. For the encoder in Figure 2.7 we obtain:

$$\mathbf{G}(D) = (1 + D + D^2 \ \ 1 + D^2) \ . \tag{2.43}$$

The encoder depicted in Figure 2.7 is only one possible realization of the generator matrix in Equation 2.43. As there is no feedback from the register outputs to the input this encoder has a finite impulse response. In general, an encoder may have feedbacks according to Figure 2.6 and therefore an infinite impulse response.

The elements of generator matrices are realizable rational functions:

$$G_{ij}(D) = \frac{P_{ij}(D)}{Q_{ij}(D)} = \frac{p_0 + p_1 D + \ldots + p_m D^m}{1 + q_1 D + \ldots + q_m D^m} \quad . \tag{2.44}$$

We call $\mathbf{G}(D)$ a *polynomial generator matrix* if $Q_{ij}(D) = 1$ for all sub-matrices $G_{ij}(D)$. Since the coefficients of a binary polynomial can be denoted by octal numbers we can achieve a shorter notation. For the generator matrix in Equation 2.43 we obtain: $\mathbf{G}(D) = (7 \;\; 5)$.

**Convolutional code, generator matrix, and encoder:** A convolutional code is the set of all possible output sequences of a convolutional encoder. We have already mentioned that a particular encoder is just one possible realization of a generator matrix. Moreover, there exists a number of different generator matrices which produce the same set of output sequences. Therefore, it is important to distinguish between properties of encoders, generator matrices, and codes. We will now give some formal definitions [JZ96].

Let $\mathbb{F}_2((D))$ denote the set of *binary Laurent series* defined by Equation 2.39. The *field of binary rational functions* $P(D)/Q(D)$, where $P(D)$ and $Q(D)$ are polynomials, is denoted by $\mathbb{F}_2(D)$ and is a subfield of $\mathbb{F}_2((D))$.

- **Generator matrix**: A generator matrix $\mathbf{G}(D)$ is a realizable $k \times n$ matrix of rank $k$.
- **Convolutional code**: A rate $R = k/n$ convolutional code $\mathcal{C}$ over $\mathbb{F}_2$ with the generator matrix $\mathbf{G}(D)$ over $\mathbb{F}_2(D)$ is the image set of the linear mapping
$$\mathbf{V}(D) = \mathbf{U}(D)\mathbf{G}(D), \;\; \mathbb{F}_2^k((D)) \rightarrow \mathbb{F}_2^n((D)) \quad .$$
- **Convolutional encoder**: A rate $R = k/n$ convolutional encoder of a convolutional code with generator matrix $\mathbf{G}(D)$ is a realization of the generator matrix by a linear sequential circuit.

**Equivalent encoding matrices:** Two encoding matrices $G(D)$ and $G'(D)$ are called *equivalent* if they encode the same code. Then

$$\mathbf{G}'(D) = \mathbf{T}(D)\mathbf{G}(D) \quad , \tag{2.45}$$

with $\mathbf{T}(D)$ a non-singular $k \times k$ matrix holds. We call $\mathbf{G}(D)$ a *systematic generator matrix* if it satisfies:

$$\begin{aligned} \mathbf{G}(D) &= \begin{pmatrix} 1 & & & G_{1,k+1}(D) & \ldots & G_{1,n}(D) \\ & 1 & & \vdots & & \vdots \\ & & \ddots & & & \\ & & 1 & \mathbf{G}_{k,k+1}(D) & \ldots & G_{k,n}(D) \end{pmatrix} \\ &= (\mathbf{I}_k \;\; \mathbf{A}(D)) \quad . \end{aligned} \tag{2.46}$$

**Finite code sequences:** In theory, the code sequences of convolutional codes are of infinite length, but for practical applications we usually employ finite sequences. There are three different methods to obtain finite code sequences:

- **Truncation:** We stop encoding after a certain number of bits without any additional efforts. This leads to high error probabilities for the last bits in a sequence.
- **Termination:** We add some tail bits to the code sequence in order to ensure a predefined end state, which leads to low error probabilities for the last bits in a sequence.
- **Tail baiting:** We choose a starting state which ensures that starting and end state are the same. This leads to equal error protection.

In general, we prefer termination or tail biting, where tail biting increases the decoding complexity and for termination additional redundancy is required. In this work we will only consider terminated code sequences, where we start encoding in the all-zero encoder state and we ensure that after the encoding process all memory elements contain zeros again. In case of a polynomial encoder this can be done by adding $k \cdot m$ zero bits to the information sequence of length $L$. As consequence we decrease the code rate

$$R_{terminated} = \frac{kL}{n(L+m)} = R\frac{L}{L+m} \ , \tag{2.47}$$

where $L/(L+m)$ is the so-called *fractional rate loss*.

### 2.3.2 Distance Properties

We will now consider some distance properties of convolutional codes. In the case of block codes we have a finite set of $2^k$ different code words and distance measures like the minimum distance are defined upon this set. With convolutional codes the situation is different, as the encoded sequences may be of infinite length, at least theoretically. Therefore, we usually consider segments of possible code sequences.

**Free Distance:** In analogy to the minimum distance of linear binary block codes we define the *free distance* of a linear binary convolutional code:

$$d_{free} = \min_{\mathbf{v},\mathbf{v}' \in \mathcal{C},\ \mathbf{v} \neq \mathbf{v}'} dist(\mathbf{v}, \mathbf{v}') \ , \tag{2.48}$$

as minimum Hamming distance between two code sequences. The free distance is a code property. Convolutional codes with the best known free distance for given rate and memory are called *optimum free distance codes* (OFD).

Before deriving further distance measures for convolutional codes it is convenient to consider a graphical representation of code sequences.

**Trellis:** A convolutional code is the set of all possible code sequences encoded by a sequential circuit, the convolutional encoder. The *physical state* of the encoder is defined as its shift register contents and will be denoted as $\sigma$. If $\mathbf{G}(D)$ is polynomial, then the dimension of the *physical state space* of its controller canonical form is equal to the over-all constraint length $\nu$. The output block of an encoder $\mathbf{v}_t$ at time $t$ is determined by the input word $\mathbf{u}_t$ and the current state $\sigma_t$.

If we want to depict code sequences we usually use a graph, the so-called *trellis*. A trellis consists of branches and nodes, where each node represents an encoder state and a branch represents a state transition. The branches are labeled with the corresponding input/output block. We order the nodes in rows and each row corresponds to a particular time step. A path in a trellis can be defined as a sequence of branches or nodes which illustrates a valid code word. In Figure 2.8 we depict the trellis for the (7 5)-convolutional code. Here each branch is labeled with the associated output symbols. A solid line represents an input one, a dashed line an input zero. In order to represent code words we only need the labeling of the output symbols. All possible paths in a trellis represent the convolutional code $\mathcal{C}$. But note, the trellis still depends on the encoder. If for a particular encoder among all possible realizations the number of physical states is minimal we call the trellis *minimal Viterbi-Forney trellis*.

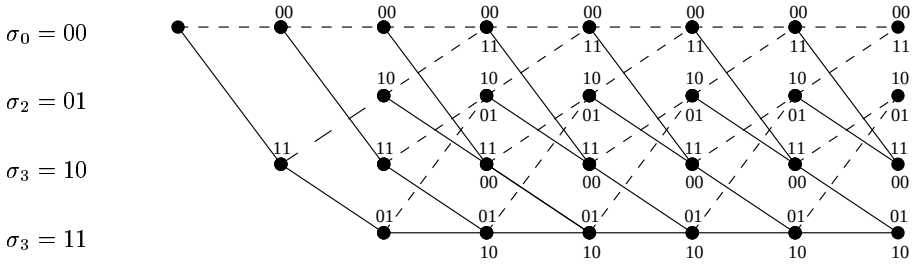



Figure 2.8: Trellis for (7 5)-convolutional code

**Active Distances:** We will now define some distance measures for convolutional codes which were recently introduced by Höst *et al.* [HJZZ98]. This family of so-called *active distances* is very useful in the analysis of concatenated convolutional codes. We will later on discuss high rate punctured convolutional codes that represent partial unit memory codes and therefore we use a definition slightly different from that in [HJZZ98] which is applicable to partial unit memory codes.

Let $\mathbf{S}_{[t_1,t_2]}^{\sigma_1,\sigma_2}$ be the set of state sequences $\sigma_{[t_1,t_2]}$ that start at depth $t_1$ in state $\sigma_1$ and terminate at depth $t_2$ in state $\sigma_2$ and do not have two consecutive zero states connected by a transition with zero output symbols in between, i.e.

$$\mathbf{S}_{[t_1,t_2]}^{\sigma_1,\sigma_2} = \left\{\sigma_{[t_1,t_2]} | \sigma_{t_1} = \sigma_1, \sigma_{t_2} = \sigma_2 \ \textit{and} \ \sigma_i = \sigma_{i+1} = 0 \ \textit{only if} \ \mathbf{v}_i \neq 0, \ t_1 \leq i < t_2\right\} \ . \tag{2.49}$$

A code word segment of length $j+1$ is denoted as $\mathbf{v}_{[t,t+j]} = v_t, v_{t+1}, \ldots, v_{t+j}$. Let $\mathcal{C}$ be a convolutional code encoded by a polynomial generator matrix $\mathbf{G}(D)$ of memory $m$ which is realized in controller canonical form. We are now prepared to define the *active distances* of $\mathcal{C}$:

- **Active row distance**: The $j$th order active row distance is

$$a^r(j) = \min_{\mathbf{s}^{0,\sigma}_{[0,j+1]},\ \sigma^{(1,i)}_{j+1+i}=0,\ 1\le j\le m} \left(wt(\mathbf{v}_{[0,j+m]})\right) \quad , \tag{2.50}$$

  where $\sigma$ denotes any value of the state $\sigma_{j+1}$ such that $\sigma^{(1)}_{j+1} \neq 0$, and $\sigma^{(1,i)}_{j+i+1}$ denotes the $i$ first positions of the shift register, i.e., $\sigma^{(1,i)}_{j+i+1} = \sigma^{(1)}_{j+i+1}, \ldots, \sigma^{(i)}_{j+i+1}$.

- **Active column distance**: The $j$th order active column distance is

$$a^c(j) = \min_{\mathbf{s}^{0,\sigma}_{[0,j+1]}} \left(wt(\mathbf{v}_{[0,j]})\right) \quad , \tag{2.51}$$

  where $\sigma$ denotes any encoder state.

- **Active reverse column distance**: The $j$th order active reverse column distance is

$$a^{rc}(j) = \min_{\mathbf{s}^{\sigma,0}_{[0,j+1]}} \left(wt(\mathbf{v}_{[0,j]})\right) \quad , \tag{2.52}$$

  where $\sigma$ denotes any encoder state.

- **Active burst distance**: The $j$th order active burst distance is

$$a^b(j) = \min_{\mathbf{s}^{0,0}_{[0,j+1]}} \left(wt(\mathbf{v}_{[0,j]})\right) \quad , \tag{2.53}$$

  where $j \geq \nu_{min}$.

- **Active segment distance**: The $j$th order active segment distance is

$$a^s(j) = \min_{\mathbf{s}^{\sigma_1,\sigma_2}_{[m,m+j+1]}} \left(wt(\mathbf{v}_{[m,j+m]})\right) \quad , \tag{2.54}$$

  where $\sigma_1$ and $\sigma_2$ denote any encoder state.

From the definitions follows that the active distances are encoder properties, not code properties. However, it also follows that this distances are invariant over the set of minimal encoding matrices[1] for a code $\mathcal{C}$. For the free distance we obtain:

$$d_{free} = \min_j(a^r(j)) \quad . \tag{2.55}$$

[1]For the definition of a minimal encoding matrix we refer to [JZ96].

In general, all active distances can be lower bounded by linear functions with the same slope $\alpha$. Therefore, we can write:

$$\begin{aligned} a^r(j) &\geq \max(\alpha \cdot j + \beta^r, d_{free}) \\ a^c(j) &\geq \alpha \cdot j + \beta^c \\ a^{rc}(j) &\geq \alpha \cdot j + \beta^{rc} \\ a^b(j) &\geq \alpha \cdot j + \beta^b \\ a^s(j) &\geq \alpha \cdot j + \beta^s \quad . \end{aligned} \tag{2.56}$$

Let $\mathcal{A}$ denote the list of parameters $(d_{free}, \alpha, \beta^b, \beta^c, \beta^{rc}, \beta^s)$, which characterizes the distance properties of a convolutional code.

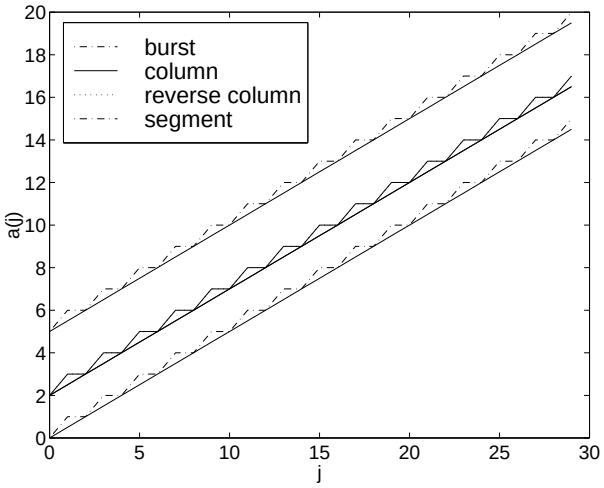



Figure 2.9: Active distances of the (7 5)-convolutional code

**Example:** *Figure 2.9 shows the active distances of the (7 5)-convolutional code, with* $\mathcal{A} = (d_{free} = 5, \alpha = 1/2, \beta^b = 9/2, \beta^c = 3/2, \beta^{rc} = 3/2, \beta^s = -1/2)$. *In Table 2.3 we present the free distances and slopes of the active distances for rate* $R = 1/2$ *OFD convolutional codes with different memories. We notice that with increasing memory the free distance improves while the slope* $\alpha$ *decreases.*

| memory | polynomial | free distance $d_{free}$ | slope $\alpha$ |
|---|---|---|---|
| 2 | (7 5) | 5 | 0.5 |
| 3 | (17 15) | 6 | 0.5 |
| 4 | (35 23) | 7 | 0.36 |
| 5 | (75 53) | 8 | 0.33 |
| 6 | (171 133) | 10 | 0.31 |

Table 2.3: Free distances and slopes $\alpha$ for rate $R = 1/2$ OFD convolutional codes with different memories

*Both parameters, $\alpha$ and $d_{free}$, characterize the distance properties of convolutional codes. It's a matter of common knowledge that the free distance determines the code performance for high signal to noise ratios. Whereas the simulation results in Figure 2.10 indicate that codes with good active distances respectively high slopes $\alpha$ achieve better results for low $E_b/N_0$ values. Latter case is important for code concatenation.*

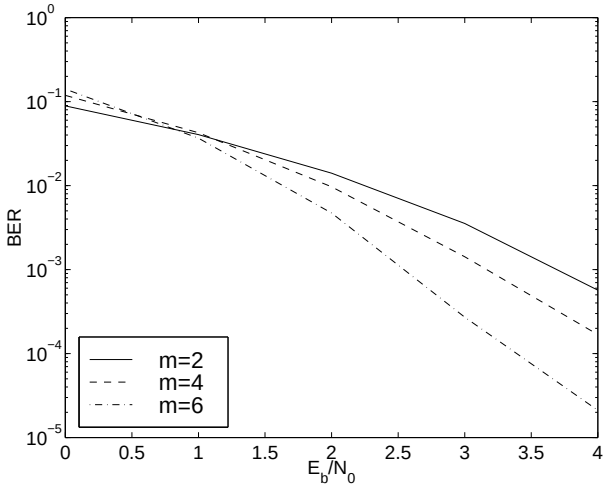



Figure 2.10: Bit error rate of convolutional codes with different memories

### 2.3.3 Puncturing Convolutional Codes

In this section we will introduce a simple method for constructing new convolutional codes from old ones, via puncturing. It is well known that maximum likelihood decoding for high-rate codes can be simplified by using punctured rate $R = 1/n$ codes. This punctured codes are obtained by periodically deleting a part of the code bits of the mother code. Usually 'good' mother codes and puncturing schemes are used, where good means that the free distance of the resulting code should be maximized. Maximum free distances are obtained by employing the best known mother code for a particular memory (OFD codes). The resulting codes are called punctured optimum free distance codes. Puncturing tables and the resulting free distances are presented e.g. in [YKH84]. We restrain this discussion to an illustrative example. A detailed description can be found e.g. in [McE96].

**Example:** *Consider the convolutional encoder shown in Figure 2.7 with generator matrix $\mathbf{G}(D) = (1 + D + D^2 \;\; 1 + D^2)$. For each encoding step the encoder produces two code bits $v_t^{(1)}$ and $v_t^{(2)}$. Now let us delete the bit $v_{2i+1}^{(2)}$ from each four-bit output block. We obtain:*

$$\mathbf{v} = (v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_1^{(2)}, v_3^{(1)}, v_3^{(2)}, \ldots) \Rightarrow \mathbf{v}_{punctured} = (v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_3^{(1)}, v_3^{(2)}, \ldots) \quad (2.57)$$

*We can express the periodic deletion pattern by a puncturing matrix, e.g:*

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad , \tag{2.58}$$

*where '0' means that the corresponding code bit will be deleted. The original rate $R = 1/2$ mother code has become a rate $R = 2/3$ code after puncturing according to $P$. The resulting generator matrix for the punctured convolutional code is derived in [McE96]:*

$$\mathbf{G}_{punctured}(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ 0 & D & 1+D \end{pmatrix} \quad . \tag{2.59}$$

## 2.4 Soft-In/Soft-Out Decoding Schemes

Up to now we have only considered encoding schemes and code properties. But the main problem in coding theory is in fact decoding. In this section we will briefly discuss two decoding algorithms implementing optimum decoding schemes. In particular we present the *Viterbi algorithm* as a maximum likelihood sequence estimator and the *BCJR algorithm* [BCJR74] which enables symbol-by-symbol maximum-a-posteriori decoding. Detailed discussions of both algorithms can be found in [Bos98].

The Viterbi and the BCJR algorithm are based on a trellis representation of the code and can by employed for the decoding of block codes and convolutional codes, but we will only consider the decoding of convolutional codes. We assume that a randomly selected information sequence $\mathbf{u}$ is encoded into a code sequence $\mathbf{v}$ and that the sequence $\mathbf{r}$ is received after transmission over a memoryless channel, in particular the AWGN channel.

In principle, we have to differentiate between optimum decisions for both code symbols and code sequences. The transmission of a code sequence over the channel changes a-priori knowledge to a-posteriori knowledge. The maximum-a-posteriori (MAP) sequence estimator is an optimum decoder selecting the sequence that maximizes the a-posteriori probability $P(\hat{\mathbf{v}}|\mathbf{r})$

$$\hat{\mathbf{v}} = \arg\left[\max_{\mathbf{v}\in\mathcal{C}} P(\mathbf{v}|\mathbf{r})\right] = \arg\left[\max_{\mathbf{v}\in\mathcal{C}} f_{\mathbf{V}|\mathbf{R}}(\mathbf{v}|\mathbf{r})\right] \quad . \tag{2.60}$$

The symbol-by-symbol maximum-a-posteriori (S/S-MAP) decoder is an optimum decoder for code symbols as it selects:

$$\hat{v}_i = \arg\left[\max_{v_i\in\mathbb{F}_2} P(v_i|\mathbf{r})\right] = \arg\left[\max_{v_i\in\mathbb{F}_2} f_{V_i|\mathbf{R}}(v_i|\mathbf{r})\right] \quad . \tag{2.61}$$

If we assume equal probable code sequences we obtain the same arguments whether we maximize $P(\hat{\mathbf{v}}|\mathbf{r})$ or $P(\mathbf{r}|\hat{\mathbf{v}})$ . A maximum likelihood (ML) decoder selects the sequence $\hat{\mathbf{v}}$ among all possible code sequences which maximizes the probability $P(\mathbf{r}|\hat{\mathbf{v}})$

$$\hat{\mathbf{v}} = \arg\left[\max_{\mathbf{v}\in\mathcal{C}} P(\mathbf{r}|\mathbf{v})\right] = \arg\left[\max_{\mathbf{v}\in\mathcal{C}} f_{\mathbf{R}|\mathbf{V}}(\mathbf{r}|\mathbf{v})\right] \quad . \tag{2.62}$$

### 2.4.1 Viterbi Decoding

A ML sequence estimator has to produce an estimate $\hat{\mathbf{v}}$ of the code word $\mathbf{v}$ based on the received sequence $\mathbf{r}$. As a consequence of the statistical independence of the additive noise symbols we obtain

$$\begin{aligned} f_{\mathbf{R}|\mathbf{V}}(\mathbf{r}|\mathbf{v}) &= \prod_{i=1}^{N} f_{\mathbf{R}_i|\mathbf{V}_i}(\mathbf{r}_i|\mathbf{v}_i) \\ &= \prod_{i=1}^{N}\prod_{j=1}^{n} f_{R|V}(r_{i,j}|v_i^{(j)}) \quad , \end{aligned} \tag{2.63}$$

where each $\mathbf{v}_i$ is a block of $n$ code bits $v_i^{(j)}$. In order to facilitate readability we often write $v_i$ instead of $v_i^{(j)}$ in the following discussion. It follows

$$\log f_{\mathbf{R}|\mathbf{V}}(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{N} \log f_{\mathbf{R}_i|\mathbf{V}_i}(\mathbf{r}_i|\mathbf{v}_i) \quad . \tag{2.64}$$

The log-likelihood function $\log f_{\mathbf{R}|\mathbf{V}}(\mathbf{r}|\mathbf{v})$ is called the *metric* associated with the path $\mathbf{v}$, and is denoted $\Lambda^{\mathbf{v}}$. With the *branch metrics* $\lambda_i^{\mathbf{v}} = \log f_{\mathbf{R}_i|\mathbf{V}_i}(\mathbf{r}_i|\mathbf{v}_i)$ we obtain

$$\begin{aligned} \Lambda_{i+1}^{\mathbf{v}} &= \Lambda_i^{\mathbf{v}} + \lambda_i^{\mathbf{v}} \\ \Lambda^{\mathbf{v}} &= \textstyle\sum_{i=1}^{N} \lambda_i^{\mathbf{v}} \quad . \end{aligned} \tag{2.65}$$

Using path metrics we are able to rewrite Equation 2.62:

$$\hat{\mathbf{v}} = \arg\left[\max_{\mathbf{v}\in\mathcal{C}} \Lambda^{\mathbf{v}}\right] \quad . \tag{2.66}$$

In order to understand Viterbi's decoding algorithm, it is convenient to consider the code trellis. As we only use terminated codes, the trellis diagram contains $N = L+m$ time steps. There are $2^{kL}$ distinct paths through the trellis corresponding to the $2^{kL}$ code words. The Viterbi algorithm searches for the path through the trellis with the largest metric. As mentioned above there are $L+m$ time steps, at each step, the algorithm compares the metrics of all paths entering each state and stores the path with the largest metric, the so-called *survivor*. See Figure 2.11. From step $m$ through step $L$ there are $2^{\nu}$ survivors per step, one for each state. At time step $L+m$, there is only one state, the all-zero state, and hence only one survivor, and the algorithm terminates.

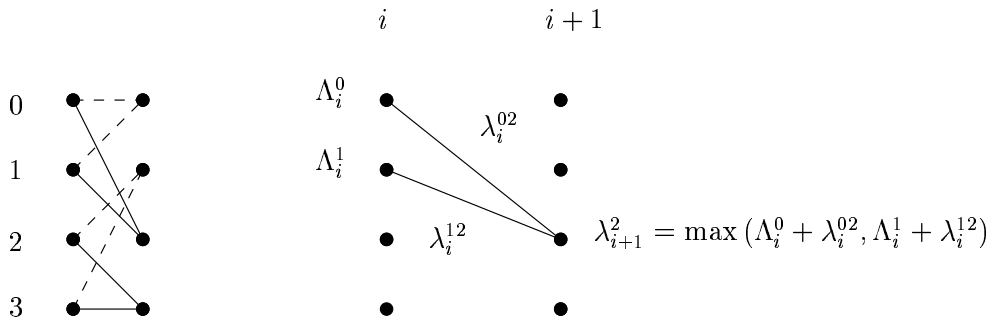



Figure 2.11: A single calculation step of the Viterbi algorithm

### 2.4.2 Symbol-by-Symbol-MAP-Decoding

Now we will discuss the BCJR algorithm which was presented by Bahl, Cocke, Jelinek and Raviv. Like the Viterbi algorithm it is implemented using code trellises, but the BCJR algorithm optimizes bit error probabilities and enables soft-output decoding which is very important for iterative decoding.

The basic idea behind the BCJR algorithm is that maximizing the a-posteriori probability $f_{V_i|\mathbf{R}}(v_i|\mathbf{r})$ is equal to maximizing the joint probability $f_{V_i\mathbf{R}}(v_i, \mathbf{r})$. Assuming a memoryless channel and independent information symbols we may represent $f_{V_i\mathbf{R}}(v_i, \mathbf{r})$ in factorized form

$$f_{V_i\mathbf{R}}(v_i, \mathbf{r}) = \underset{\substack{\uparrow \\ backward}}{f_{\mathbf{R}_i^B|V_i}(\mathbf{r}_i^B|v_i)} \cdot \underset{\substack{\uparrow \\ current}}{f_{V_iR_i}(v_i, r_i)} \cdot \underset{\substack{\uparrow \\ forward}}{f_{\mathbf{R}_i^F|V_i}(\mathbf{r}_i^F|v_i)} \tag{2.67}$$

The first term is the probability of $v_i$ conditioned on the first $i-1$ terms of $\mathbf{R}$ (backward part). The second term is the joint probability of $v_i$ and $i$th received symbol $r_i$. And finally the last term is the probability of $v_i$ conditioned on the remaining $N-i$ terms of $\mathbf{R}$ (forward part). These probabilities can be evaluated using the Viterbi algorithm.

Introducing a metric we are able to evaluate Equation 2.61 with the symbol-by-symbol BCJR algorithm. The algorithm passes through the trellis twice. During the backward recursion (Viterbi algorithm from the right end of the trellis to the left) we calculate the transition probabilities $\lambda_i$ and the accumulated probabilities $\Lambda_i^B$ for each node in the trellis

$$\Lambda_i^B = \Lambda_{i+1}^B + \lambda_i \ . \tag{2.68}$$

In the second procedure step, the forward recursion, we calculate the accumulated probabilities $\Lambda_i^F$ from left to right

$$\Lambda_{i+1}^F = \Lambda_i^F + \lambda_i \tag{2.69}$$

and finally we evaluate

$$\hat{v}_i = \arg \left[ \max_{v_i \in \mathbb{F}} \Lambda_{i+1}^B + \lambda_i + \Lambda_i^F \right] \quad . \tag{2.70}$$

Note: In order to simplify the description of the BCJR algorithm we have neglected the fact that a transition in the trellis depends on $n$ code bits and the a-priori probabilities of the corresponding $k$ information symbols. Above metric is an approximation which considers only the most probable transition (Max-Log-MAP algorithm). For a detailed description of the algorithm we refer to [Bos98] and [SB98].

**Example:** *Here we present some simulation results for a memory $m = 6$ rate $R = 1/2$ convolutional code. We compare two different encoding schemes: recursive systematic and polynomial non-systematic encoding. Although both encoding schemes encode the same code we observe that for high signal to noise ratios non-systematic encoding yields a better performance. On the other hand for low signal to noise ratios we observe a reciprocal result.*

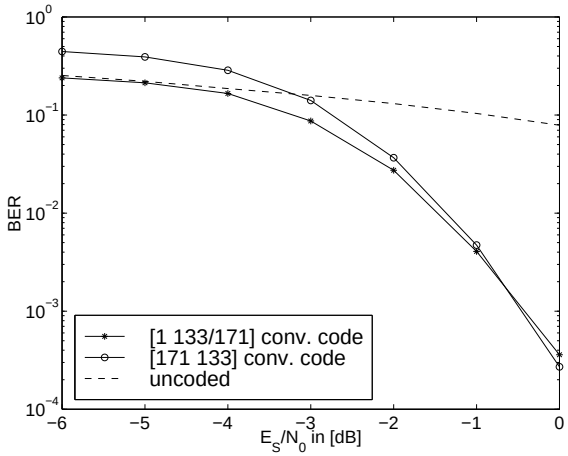



Figure 2.12: S/S-MAP decoding of a systematic and non-systematic encoded codes

## 2.4.3 Soft-In/Soft-Out Decoding

Soft-output decoding generally consists of a hard decision and a measure for the probability that this decision is correct which is often called reliability. For applications where soft-in and soft-out values are required the use of log-likelihood ratios as metric is often favorable.

Let $X$ be a binary random variable from an alphabet $\{-1, +1\}$. The log-likelihood ratio of $X$ is defined as

$$L_X(x) = \ln \frac{P(X = +1)}{P(X = -1)} \quad . \tag{2.71}$$

We call $L_X(x)$ the L-value of $X$, where the sign of $L_X(x)$ corresponds to the hard decision and the magnitude $|L_X(x)|$ represents the reliability of the decision. We often skip the indices for the random variable and simply write $L(x)$.

Consider the BCJR algorithm. We may express the joint probability $f_{V_i\mathbf{R}}(v_i, \mathbf{r})$ as the product of three terms:

$$f_{V_i\mathbf{R}}(v_i, \mathbf{r}) = \underset{\substack{\uparrow \\ channel}}{f_{R_i|V_i}(r_i|v_i)} \cdot \underset{\substack{\uparrow \\ a-priori}}{f_{V_i}(v_i)} \cdot \underset{\substack{\uparrow \\ extrinsic}}{f_{\tilde{\mathbf{R}}|V_i}(\tilde{\mathbf{r}}|v_i)} \quad , \tag{2.72}$$

where the first term represents the conditional probabilities describing the memory less channel, the second term is the a-priori probability of the code symbol, and, last but not least, the so-called extrinsic part from code correlations. We will later on explain an iterative decoding algorithm for concatenated codes. With iterative decoding we only use the extrinsic part of the a-posteriori knowledge of a decoding step to gain new a-priori knowledge for the next iteration.

Let us consider an AWGN channel with variance $\sigma_n^2 = 1/2N_0$ and BPSK modulation. Using L-values it follows

$$L_{R|V}(r|v) = \ln \frac{\exp\left(-\frac{1}{2\sigma_n^2}(r - E_s)^2\right)}{\exp\left(-\frac{1}{2\sigma_n^2}(r + E_s)^2\right)} = L_{ch} \cdot r \quad , \tag{2.73}$$

with $L_{ch} = 4E_s/N_0$. For the a-priori probability we obtain

$$L_U(u) = \ln \frac{f_U(u = +1)}{f_U(u = -1)} = L_a(u) \quad . \tag{2.74}$$

In general, a soft-in/soft-out (SISO) decoder, such as an implementation of the BCJR algorithm, may expects channel L-values and a-priori information as an input and may produce reliabilities for the estimated information symbols $L(\hat{u})$, extrinsic L-value for code $L_e(\hat{v})$ and information bits $L_e(\hat{u})$ (see Figure 2.13).


Figure 2.13: Soft-In/Soft-Out decoder and L-values

Consider a rate $R = 1/2$ systematic encoded convolutional code with generator matrix $G(D) = (1 \;\; A(D))$, then for the estimation of information bits we obtain $L(\hat{u_i}) = L(\hat{v_{2i}})$. The BCJR algorithm employing L-values delivers soft-output values

$$L(\hat{u_i}) = L_a(u_i) + L_{ch} \cdot r_{2i} + L_e(\hat{u_i}) \quad . \tag{2.75}$$

It follows that the extrinsic reliabilities can simply be calculated by

$$L_e(\hat{u_i}) = L(\hat{u_i}) - L_a(u_i) - L_{ch} \cdot r_{2i} \quad . \tag{2.76}$$

# Chapter 3

# Parallel and Serial Concatenation

But note, the decoding of concatenated codes may not be ML decoding. In this chapter we will discuss parallel (PCC) and serial concatenated codes (SCC).

*Turbo codes* are a class of error correcting block codes, based on a parallel concatenated coding scheme, where at least two systematic encoders are linked by an interleaver. In the original paper [BGT93], Berrou *et al.* showed by simulation that turbo codes employing convolutional component codes are capable of achieving bit error rates as small as $10^{-5}$ at a code rate of $R = 1/2$ and a $E_b/N_0$ ratio of just 0.7 dB above the theoretical *Shannon limit*. However, in order to achieve this level of performance, large block sizes of 65,532 code bits are required. The name *turbo* reflects a property of the employed iterative decoding algorithm: The decoder output of one iteration is used as decoder input of the next iteration.

In order to improve the performance of parallel concatenated codes Benedetto *et al.* applied these decoding scheme to serial concatenated codes with interleaving [BM98]. The corresponding encoders consist of the cascade of an outer encoder, an interleaver permuting the outer code word, and an inner encoder whose input words are the permuted outer code bits.

We will deal with both constructions, starting by presenting the basic concepts of parallel and serial concatenation and then applying these concepts to constructions with interleaving. Although there are many publications on turbo codes the theoretical description of this codes is still difficult. A lot of questions concerning code performance, decoding, and code design remain unanswered.

# 3.1 Basic Concepts of Parallel and Serial Concatenations

## 3.1.1 Parallel and Serial Encoding Schemes

In Figure 3.1 we see the code word of a parallel concatenation of two block codes. Both component codes $\mathcal{C}_1(n_1, k_1)$ and $\mathcal{C}_2(n_2, k_2)$ are encoded by a systematic generator matrix $\mathbf{G}_1$ respectively $\mathbf{G}_2$:

$$\mathbf{G}_i = [\mathbf{I}_k \;\; \mathbf{A}_i] \quad . \tag{3.1}$$

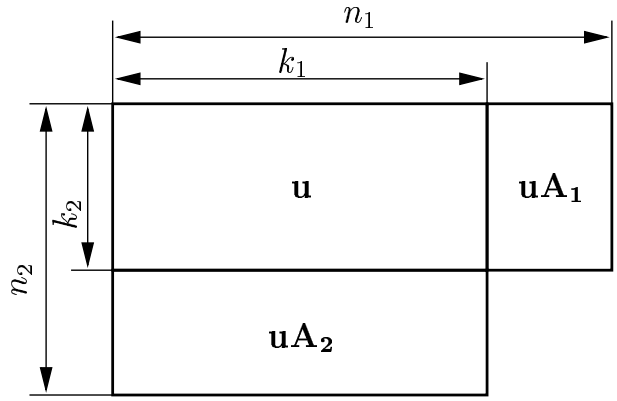



Figure 3.1: Parallel Concatenated Code

The information sequence is split into information words of $k = k_1 \cdot k_2$ bits, which are written into a $k_1 \times k_2$ matrix. We first encode row-wise to generate the parity bits for $\mathcal{C}_1$. Then we encode column-wise and obtain the parity bits for $\mathcal{C}_2$. The information part is only transmitted once. The rate of the over-all code is:

$$R = \frac{k}{n} = \frac{k_1 k_2}{k_1 k_2 + k_1(n_2 - k_2) + k_2(n_1 - k_1)} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} - 1} \quad . \tag{3.2}$$

For the minimum distance of the concatenation we obtain:

$$d_{min} \geq \max\{d_1, d_2\} \quad , \tag{3.3}$$

with $d_1$ and $d_2$ the minimum distances of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively.

In Figure 3.2 we depict the code word of a serial concatenation of two systematic encoded block codes. The over-all code rate is

$$R = R_1 \cdot R_2 \quad . \tag{3.4}$$

For the minimum distance of the serial concatenation we can achieve

$$d_{min} \geq d_1 \cdot d_2 \quad , \tag{3.5}$$

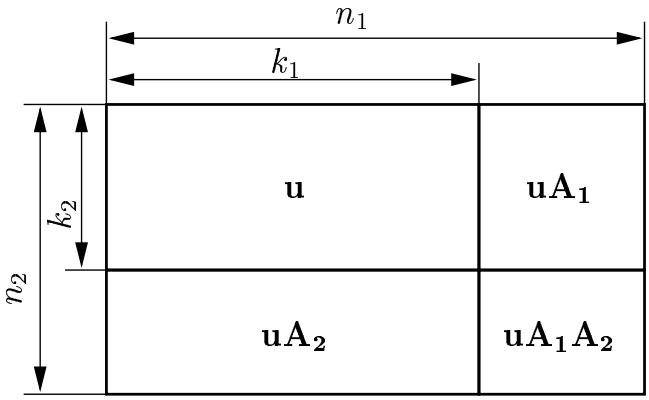



Figure 3.2: Serial Concatenated Code

if the serial concatenation is a product code. Thus, the minimum distance of parallel concatenated codes is usually smaller than the minimum distance of serial concatenations with the same component codes. But on the other hand, we obtain a gain in code rate. If we assume two identical component codes in both schemes, this gain in decibels is given by $R_{gain} = -10\log_{10}(R_c(2 - R_c))$, where $R_c$ denotes the code rate of the component codes [LB98]. In Figure 3.3 the code rate gain is plotted and we observe that this gain decreases with increasing component code rate.

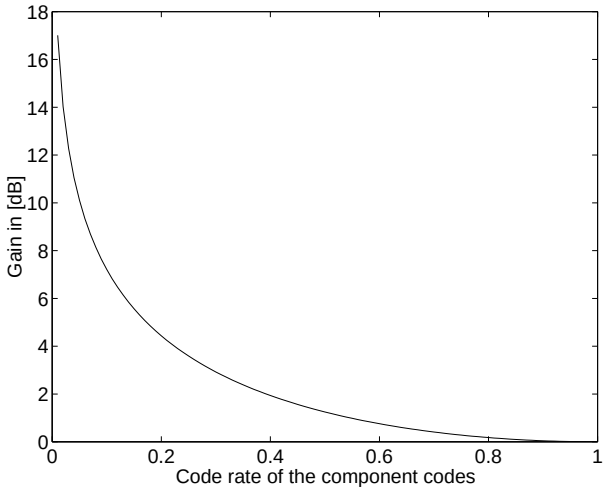



Figure 3.3: Code rate gain of a parallel concatenated code in comparison to serial concatenation

### 3.1.2 Iterative Soft-In/Soft-Out Decoding

Before extending these encoding concepts to more sophisticated codes like turbo codes we will discuss the decoding scheme. In fact, the great success of turbo codes

is mainly based on an iterative scheme employing soft-in/soft-out decoding (SISO-decoding) of the component codes. Although iterative decoding can be used in a more general way than just for the decoding of parallel concatenated codes, e.g. iterative channel equalization and channel decoding, it is often denoted as *turbo decoding* [Hag97].

In terms of code word error probability, maximum likelihood sequence estimation is the optimum decoding principle, provided that all code words are equiprobable. But the computational complexity of optimum decoding increases exponentially with the dimension of block codes or with the constraint length of convolutional codes. As a consequence, its practical application is limited to codes of moderate dimension or memory. But as we know from information theory, increasing code word or constraint length leads to better performance. Consequently, the aim from the viewpoint of decoding is the development of decoding algorithms with low complexity which allow the decoding of long powerful codes.

The iterative decoding algorithm can be used for decoding long codes obtained from concatenated codes. Here the decoding is split into several steps and symbol-by-symbol reliability information is repeatedly transferred between the decoding steps. The complexity of the iterative algorithm depends on the complexity of the SISO-decoding of the component codes.

In the following discussion of the turbo decoding algorithm we use *log-likelihood ratios* (L-values) and assume symbol-by-symbol MAP decoding for the decoding of the component codes. The low complexity of SISO-decoding of convolutional codes is a main reason for using convolutional codes in concatenated systems.

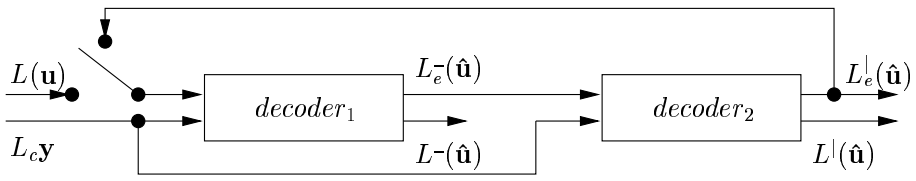



Figure 3.4: Iterative decoding scheme

A general iterative decoding scheme is depicted in Figure 3.4. Each SISO-decoder accepts a-priori and channel L-values and delivers soft-output L-values

$$L(\hat{\mathbf{u}}) = L_{ch} \cdot \mathbf{r} + L_a(\mathbf{u}) + L_e(\hat{\mathbf{u}}) \ , \tag{3.6}$$

where we assume systematic encoded component codes. The soft output information of each information symbol consists of three parts: Reliabilities from the channel, the a-priori knowledge, and the so-called *extrinsic* part from code correlations. Only the extrinsic values should be used to gain the new a-priori values for the next iteration. At least during the first iteration these values are statistically independent.

We presume equal probable information bits, thus $L(\mathbf{u})$ equals zero for the first decoding step. The $decoder_1$ calculates $L^{-}(\hat{\mathbf{u}})$ and $L_e^{-}(\hat{\mathbf{u}})$ from the soft-in channel information and passes the extrinsic values as a-priori information to $decoder_2$. The second decoder feeds $L_e^{|}(\hat{\mathbf{u}})$ back to $decoder_1$ as a-priori information for the next iteration, and so on. After the final iteration the reliabilities $L^{|}(\hat{\mathbf{u}})$ are the soft-out values of the *iterative soft-in / soft-out decoder*.

## 3.2 Encoding and Decoding of Turbo Codes

To start with turbo codes, it is convenient to consider their encoding scheme. As mentioned above, turbo codes result from a parallel concatenation of a number of systematic encoders linked by interleavers. Although the component codes may be convolutional codes the resulting turbo code is a block code. This follows from the block-wise encoding scheme and the fact that convolutional component codes have to be terminated. The general encoding scheme for turbo codes is depicted in Figure 3.5.

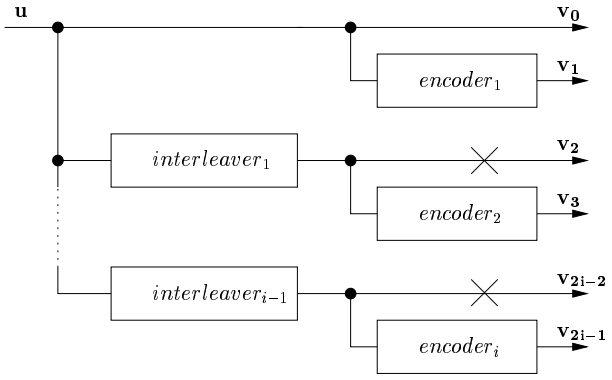



Figure 3.5: Encoder scheme for turbo codes

The information bits are interleaved after encoding the first component code. As we see, the parallel concatenation scheme can easily be generalized to more than two component codes. If not mentioned otherwise, we will only consider turbo codes with two equal component codes in the following discussion. The rate of such a code is:

$$R = \frac{R_c}{2 - R_c} \quad , \tag{3.7}$$

where $R_c$ is the rate of the component codes. The code word of the resulting code has the following structure:

$$\mathbf{v} = [\ \mathbf{u}\ |\ \mathbf{u}\mathbf{A}_1\ |\ \pi(\mathbf{u})\mathbf{A}_2\ ] \quad , \tag{3.8}$$

where $\pi(\cdot)$ denotes a permutation of the information bits and $\mathbf{A}_i$ denotes a submatrix of the systematic encoding matrix $G_i = [I_k\ \ A_i]$.

To apply the iterative decoding scheme for decoding of turbo codes we have to adapt it according to Figure 3.6. Here, $\pi, \pi^{-1}$ denote the interleaver respectively deinterleaver. The channel input of the decoder is split into three parts: Channel L-values from information bits, parity bits belonging to the first code, and parity bits belonging to the second code.

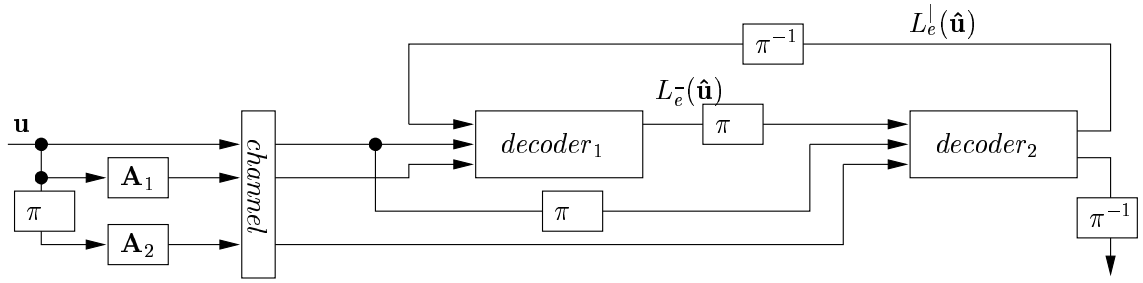



Figure 3.6: Decoder scheme for turbo codes

## 3.3 Serial Concatenated Convolutional Codes

We will now consider the encoding and decoding of serial concatenated convolutional codes with interleaving as introduced in [BM98]. The encoding scheme as depicted in Figure 3.7 is quite simple. The encoder consists of the cascade of an outer convolutional encoder, an interleaver permuting the outer code word, and an inner convolutional encoder encoding the permuted outer code bits.

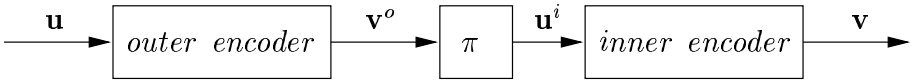



Figure 3.7: Encoder scheme for serial concatenated convolutional codes with interleaving

The decoding scheme presented in Figure 3.8 is similar to the iterative decoding scheme for turbo codes. The decoder for the inner code generates reliabilities $L(\hat{\mathbf{u}}^i)$ which represent the permuted channel values of the outer decoder. The outer decoder calculates L-values for the information word $L(\hat{\mathbf{u}})$ needed for the final decision and additionally provides extrinsic values $L_e(\hat{\mathbf{v}}^o)$ for outer code bits. These extrinsic values are interleaved and treated as a-priori information for the next decoding step.

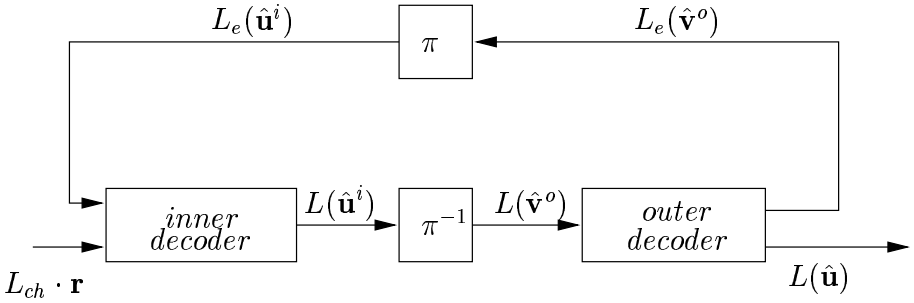



Figure 3.8: Decoding scheme for serial concatenated convolutional codes with interleaving

## 3.4 Theoretical Analysis

Since the proposal of turbo codes, neither a good theoretical explanation of the code performance nor an adequate comprehension considering the role of the component codes and the interleaving appeared. Thus, most we know about turbo codes is based on simulation results. We will analyse some code properties of turbo codes and serial concatenated convolutional codes. This discussion will mainly be based on simulations, but in addition we will compare the simulation results with some theoretical considerations from [Ben96], [BM98] and [Svi95].

### 3.4.1 Weight Distribution and Bounds

Although the union bound is based on the assumption of maximum likelihood decoding it is often used to explain the performance of turbo codes. In order to facilitate the following discussion we repeat Equation 2.31 and 2.34 from Chapter 2.

- union bound on word error probability:

$$P_W \leq \sum_{w=d_{min}}^{n} A_w \cdot P(w) \ . \tag{3.9}$$

- union bound on bit error probability:

$$P_B \leq \sum_{w=d_{min}}^{n} B_w \cdot P(w) \ . \tag{3.10}$$

Consider an AWGN channel, then $P(w) = \frac{1}{2} erfc\left(\sqrt{w\frac{E_s}{N_0}}\right)$ is the probability that an error pattern of weight $w$ occurs. $A_w$ denotes the coefficients of the WEF and

$B_w$ represents the bit error multiplicities. Due to the quick decrease of $P(w)$ with growing over-all weight $w$ we conclude that the Hamming weight of all code words other then the all zero one should be as high as possible.

Now let us consider the case of turbo codes. The over-all Hamming weight of a code word is the sum over the Hamming weights of the three code word parts: Information bits, first and second parity part. Code words with low weight in the first redundancy part should be associated by interleaving with a second redundancy part with high weight and vice versa. It is obvious that for two identical component codes this is not true without interleaving. In the original paper [BGT93], a random permutation of the information bits was employed. An examination of the literature shows that this is a widely spread approach for interleaver design. But there is also evidence that we can do better [DZ97].

There exist $k!$ different interleavers with interleaver length $k$. Investigating the complete set of possible interleavers soon becomes exceedingly complicated. We will now present two theoretical approaches to overcome this problem. Svirid obtained an 'optimistic' WEF by introducing *fully optimal interleaving* in [Svi95]. In order to achieve the highest possible minimum distance we sub-divide the set of all possible code words into $k$ groups. The $i$th group consists of all $\binom{k}{i}$ code words with weight $i$ in the information part. Let $j(i,l)$ denote the parity weight corresponding to the $l$th code word in the $i$th group. Now we organize the code words in each group according to the weight of the redundancy part so that for any $l$ holds: $j(i,l+1) \geq j(i,l)$. A fully optimal interleaver provides the code word with lowest weight in the first parity part being mapped to the second parity part with highest weight and so on. See Figure 3.9. A fully optimal interleaver is a theoretical device and Svirid proved that it is not realizable. But we can easily obtain the WEF of turbo codes employing fully optimal interleaving.

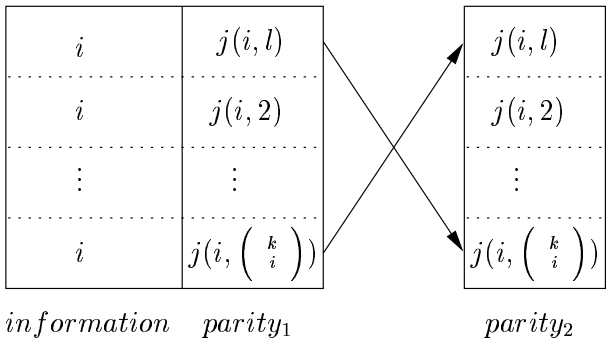



Figure 3.9: Fully optimal interleaving

Benedetto also introduced an abstract interleaver in [Ben96], called *uniform interleaver*. A uniform interleaver of length $k$ is a device which maps a given input word of weight $i$ into all distinct $\binom{k}{i}$ permutations of it with equal probability. An example for uniform interleaving is depicted in Figure 3.10. Benedetto proved that, if we assume uniform interleaving, we can easily obtain the expectation of

the conditional weight enumerating function $A^{PCC}(i, J)$ resulting from a parallel concatenation by

$$E[A^{PCC}(i, J)] = \frac{A^{(1)}(i, J) \cdot A^{(2)}(i, J)}{\binom{k}{i}} \quad . \tag{3.11}$$

$A^{(1)}(i, J)$ and $A^{(2)}(i, J)$ are the conditional weight enumerating functions of the IRWEF corresponding to the component encoders.

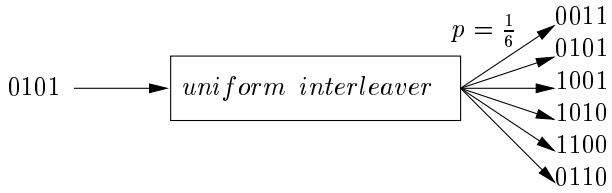



Figure 3.10: Example for uniform interleaving

In comparison to fully optimal interleaving Benedetto's approach is more practical as we can estimate uniform interleaving with our simulations, e.g. by randomly choosing a new interleaver for each transmitted block.

The expectation of the conditional weight enumerating function for serial concatenated codes with interleaving is presented in [BM98]. Let $A^o(I, W)$ denote the IOWEF of the outer encoder and $A^i(I, W)$ the IOWEF of the inner encoder, respectively. Assuming *uniform interleaving* as introduced in [Ben96] we are now able to evaluate the expectation of the weight enumerating function $A^{SCC}(I, W)$:

$$E[A^{SCC}(I, W)] = \sum_{l=0}^{N_o} \frac{A^o(I, l) \cdot A^i(l, W)}{\binom{N_o}{l}} \quad , \tag{3.12}$$

with $N_o = (K_o + m_o k_o)/R_o$, and $K_o$, the total number of information bits per transmitted block.

**Example (from [Ben96] and [BM98]):** *Consider the* $(7, 4)$ *Hamming code with the following input-redundancy weight enumerating function:*

$$A^H(I, J) = 1 + I(3J^2 + J^3) + I^2(3J + 3J^2) + I^3(1 + 3J) + I^4 J^3$$

*First, we will calculate the expectation of the IRWEF of a parallel concatenation with two identical Hamming component codes.*

$$\begin{aligned}
E[A^{PCC}(i, J)] &= \frac{A^H(i,J) \cdot A^H(i,J)}{\binom{4}{i}} \\
E[A^{PCC}(0, J)] &= \frac{1}{1} \\
E[A^{PCC}(1, J)] &= \frac{(3J^2+J^3)^2}{4} \\
E[A^{PCC}(2, J)] &= \frac{(3J^2+3J^3)^2}{6} \\
E[A^{PCC}(3, J)] &= \frac{(1+3J)^2}{4} \\
E[A^{PCC}(4, J)] &= \frac{(J^3)^2}{1}
\end{aligned}$$

*And obtain:*

$$\begin{aligned} E[A^{PCC}(I,J)] &= 1 + I(2.25J^4 + 1.5J^5 + 0.25J^6) + \\ &\quad + I^2(1.5J^2 + 3J^3 + 1.5J^4) + I^3(0.25 + 1.5J + 2.25J^2) + I^4J^6 \end{aligned}$$

*Next, we calculate the expectation of the IOWEF of a serial concatenation with an outer parity-check code $\mathcal{C}_{PC}(4,3)$ and an inner Hamming component code $\mathcal{C}_H(7,4)$. Here the input-output weight enumerating functions of the component codes are required:*

$$\begin{aligned} A^{PC}(I,W) &= 1 + I(3W^2) + I^2(3W^2) + I^3(W^4) \\ A^{H}(I,W) &= 1 + I(3W^3 + W^4) + I^2(3W^3 + 3W^4) + I^3(W^3 + 3W^4) + I^4W^7 \ . \end{aligned}$$

*We obtain:*

$$\begin{aligned} E[A^{SCC}(I,W)] &= \sum_{l=0}^{4} \frac{A^{PC}(I,l) \cdot A^{H}(l,W)}{\binom{4}{l}} \\ &= \frac{1}{1} + \frac{(3I + 3I^2)(3W^3 + 3W^4)}{6} + \frac{I^3W^7}{1} \\ &= 1 + I(1.5W^3 + 1.5W^4) + I^2(1.5W^3 + 1.5W^4) + I^3W^7 \ . \end{aligned}$$

### 3.4.2 Evaluating the WEF of Convolutional Codes

In order to calculate the expectations $E[A^{PCCC}(I,J)]$ and $E[A^{SCCC}(I,W)]$ of parallel and serial concatenations with convolutional component codes (PCCC respectively SCCC) we need the IRWEF or IOWEF of the particular component encoders. We will now describe an algorithm to evaluate weight distributions of terminated convolutional codes. A similar method is presented in [And96].

Consider the trellis of a convolutional encoder with over-all constraint length $\nu$ and a physical state space $\{\sigma_0, \ldots, \sigma_{2^\nu - 1}\}$. We define a $2^\nu \times 2^\nu$ transition matrix $\mathbf{T}$. The coefficient $\tau_{i,j}$ of $\mathbf{T}$ is the weight enumerator of the transition from state $\sigma_i$ to state $\sigma_j$, where we obtain $\tau_{i,j} = 0$ for impossible transitions.

We only consider terminated codes with $k \cdot L$ information and $n \cdot (L + m)$ code bits, thus the trellis diagram contains $N = L + m$ time steps. For the WEF we obtain

$$A(W) = (1 \ \ 0 \ \ \ldots \ \ 0) \cdot \begin{pmatrix} \tau_{0,0} & \cdots & \tau_{1,1} \\ \vdots & & \vdots \\ \tau_{0,2^\nu-1} & \cdots & \tau_{2^\nu-1,2^\nu-1} \end{pmatrix}^N \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \ , \tag{3.13}$$

which we may evaluate iteratively

$$\begin{aligned} \mathbf{A}_0(W) &= (1 \ \ 0 \ \ \ldots \ \ 0)^T \ , \\ \mathbf{A}_{i+1}(W) &= \mathbf{T} \cdot \mathbf{A}_i(W) \\ A(W) &= (1 \ \ 0 \ \ \ldots \ \ 0) \cdot \mathbf{A}_N(W) \ . \end{aligned} \tag{3.14}$$

**Example:** *As an example we depict the trellis of the (7 5)-convolutional code in Figure 3.11, which consists of four states and eight possible branches per time step. Each branch is labeled with a branch enumerator, e.g. the transition from the all-zero state to state $\sigma_2$ corresponds to an output weight two and therefore we obtain $\tau_{0,2} = W^2$. We obtain:*

$$\mathbf{A}_{i+1}(W) = \begin{pmatrix} A_{i+1}^0(W) \\ A_{i+1}^1(W) \\ A_{i+1}^2(W) \\ A_{i+1}^3(W) \end{pmatrix} = \begin{pmatrix} 1 & W^2 & 0 & 0 \\ 0 & 0 & W & W \\ W^2 & 1 & 0 & 0 \\ 0 & 0 & W & W \end{pmatrix} \cdot \begin{pmatrix} A_i^0(W) \\ A_i^1(W) \\ A_i^2(W) \\ A_i^3(W) \end{pmatrix}, \quad \mathbf{A}_0(W) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

*Now we consider the procedure for $L = 2$:*

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{i=1} \begin{pmatrix} 1 \\ 0 \\ W^2 \\ 0 \end{pmatrix} \xrightarrow{i=2} \begin{pmatrix} 1 \\ W^3 \\ W^2 \\ W^3 \end{pmatrix} \xrightarrow{i=3} \begin{pmatrix} 1+W^5 \\ W^3+W^4 \\ W^2+W^3 \\ W^3 \end{pmatrix} \xrightarrow{i=4} \begin{pmatrix} 1+2W^5+W^6 \\ W^3+2W^4+W^5 \\ W^2+W^3+W^4+W^7 \\ W^3+2W^4+W^5 \end{pmatrix},$$

*which results in the weight enumerating function*

$$A(W) = A_{L+m=4}^0(W) = 1 + 2W^5 + W^6$$

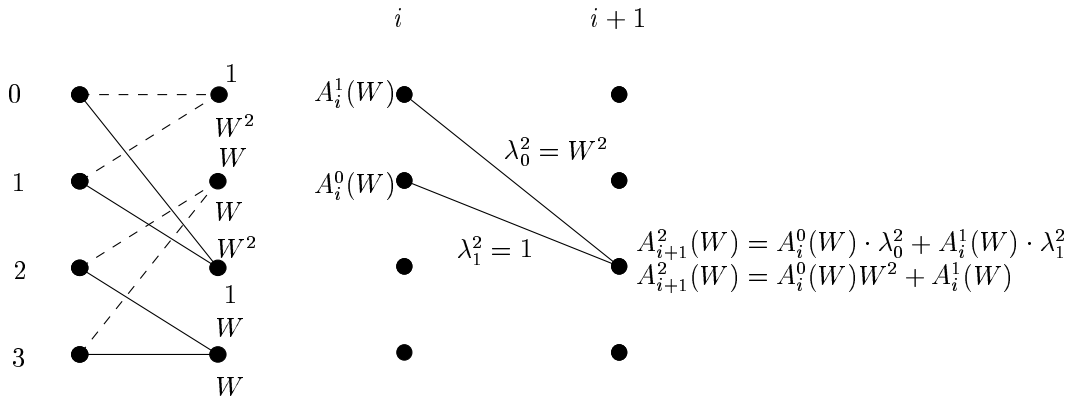



Figure 3.11: Trellis with branch enumerators

To evaluate input-output or input-redundancy weight enumerating functions two transition matrices $\mathbf{T}$ and $\mathbf{T}'$ are required. The coefficients $\tau_{i,j}$ of $\mathbf{T}$ are input-output respectively input-redundancy weight enumerators, for example $\tau_{0,2} = IW^2$. The matrix $\mathbf{T}'$ regards the tail bits necessary for termination and therefore contains only enumerators for code or redundancy bits, e.g. $\tau'_{0,2} = W^2$. We obtain

$$A(W) = (1 \;\; 0 \;\; \ldots \;\; 0) \cdot T^L T'^m \cdot (1 \;\; 0 \;\; \ldots \;\; 0)^T \quad . \tag{3.15}$$

### 3.4.3 Examples

We have already mentioned that the WEF can be used to upper bound the bit error rate employing the union bound. But the calculation complexity to obtain the complete weight distribution of a particular code increases exponentially with growing code dimension. To overcome this problem we do not investigate the actual WEF but the expectations $E[A^{PCCC}(I,J)]$ respectively $E[A^{SCCC}(I,W)]$. Note: In order to derive the expectations we assumed uniform interleaving, thus we neglect the influence of the particular interleaver.

Before considering expectations, we will compare concatenations employing different interleavings. Consider rate $R = 1/2$ turbo codes of dimension $k = 16$ with two punctured (1 5/7)-convolutional codes as component codes. In Figure 3.12 we depict the actual weight distribution of two turbo codes with different interleavers and of the terminated (1 5/7) mother convolutional code. We observe in the left graph that the over-all distributions hardly differ.

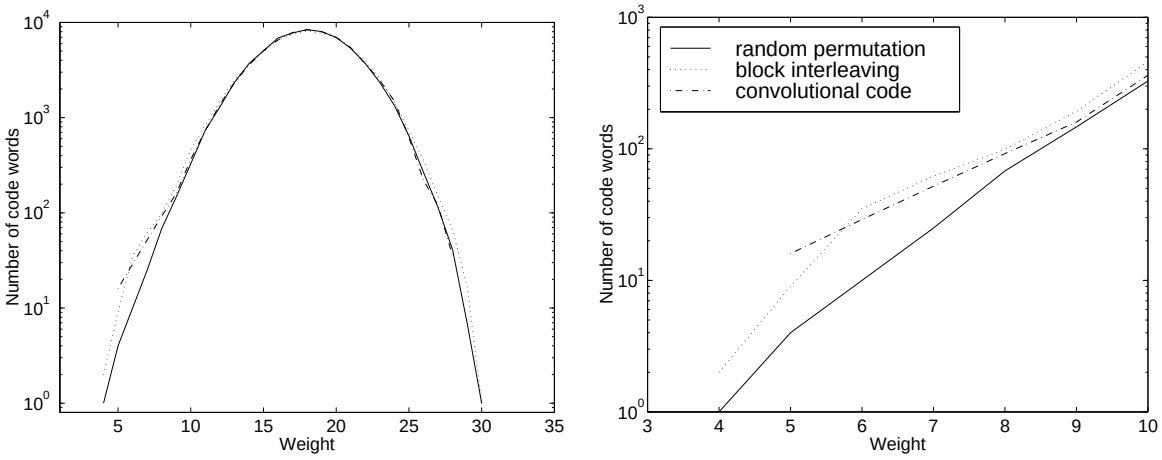



Figure 3.12: Weight distribution of a turbo code

On the right hand side of Figure 3.12 we depict only the number of code words with low Hamming weight. Here a significant difference is obvious. The turbo code with random interleaver has a lower minimum distance than the convolutional mother code, but the number of code words with small weights is much lower. In order to obtain a turbo code with a code rate equal to the rate of the mother code, we have to puncture the component codes. Therefore, the free distance of the component codes is usually smaller then $d_{free}$ of the mother code, in the example $d_1 = d_2 = 2$. We notice that the WEF and therefore the performance of a turbo code depends on the interleaving.

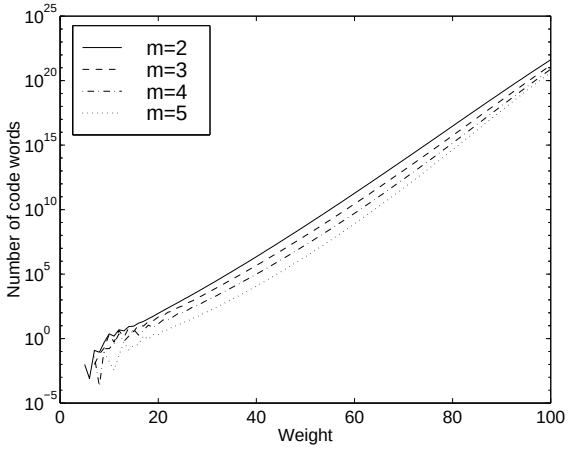
Table 3.1 contains the number of code words with low weight and the expectation for uniform interleaving. We observe that the WEF of the turbo code with randomly chosen interleaver is very close to the expectation for uniform interleaving. From

the expectation we conclude that there exist interleavings which lead to minimum distance $d_{min} = 2$, but we can probably find turbo codes with higher minimum distances than with our particular random interleaver. The $4 \times 4$ block interleaver leads to the same minimum distance but to a WEF with more low weight code words and therefore worse performance. Comparing the turbo code with the convolutional mother code we observe a lower minimum distance but fewer low weight code words.

| weight | uniform interleaver | random interleaver | block interleaver | convolutional code |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0.1 | 0 | 0 | 0 |
| 3 | 0.4 | 0 | 0 | 0 |
| 4 | 1.6 | 1 | 1 | 0 |
| 5 | 3.7 | 4 | 7 | 16 |
| 6 | 7.4 | 7 | 22 | 29 |
| 7 | 13.9 | 32 | 49 | 52 |

Table 3.1: Number of code words with weight 1 to 7

We will now investigate the influence of the component codes. Figure 3.13 displays the WEF of rate $R = 1/3$ and $k = 100$ turbo codes with different component codes. We observe that component codes with higher memory achieve fewer low weight code words. As we will see later on in the simulation results, the performance of turbo codes does not necessarily improve by choosing component codes with higher memory.

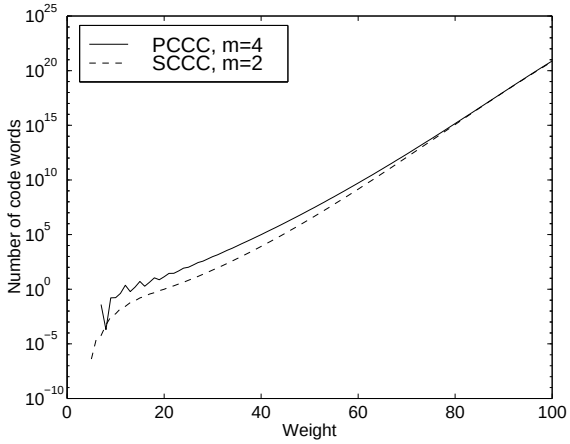



- rate $R = 1/3$
- dimension $k = 100$

Figure 3.13: Weight distribution of turbo codes with different component codes

In Figure 3.14 we compare the WEF of a turbo code with the weight distribution of a serial concatenated convolutional code. Here we notice that the SCCC with lower component code memory achieves an even better distribution, but again we will observe in Chapter 5 that rate $R = 1/3$ turbo codes usually perform better, at least

for low signal to noise ratios. Both observations may be due to the sub-optimum iterative decoding algorithm.




- rate $R = 1/3$
- dimension $k = 100$
- $m_{PCCC} = 4$
- $m_{SCCC} = 2$

Figure 3.14: Weight distribution of a turbo code and an SCCC

# Chapter 4

# Woven Codes

In this chapter we will discuss a new class of serial concatenated codes, the so-called *woven codes*. *Binary woven convolutional codes* were introduced by Höst *et al.* in [HJZ97]. With this construction several convolutional encoders are *woven* together in a manner that resembles the structure of a fabric. We will see that this construction achieves large free distances. Moreover, we will introduce and discuss some new encoder constructions: Woven encoders with additional interleaving between outer and inner encoders, where we employ terminated convolutional codes as component codes.

## 4.1 Encoding Scheme

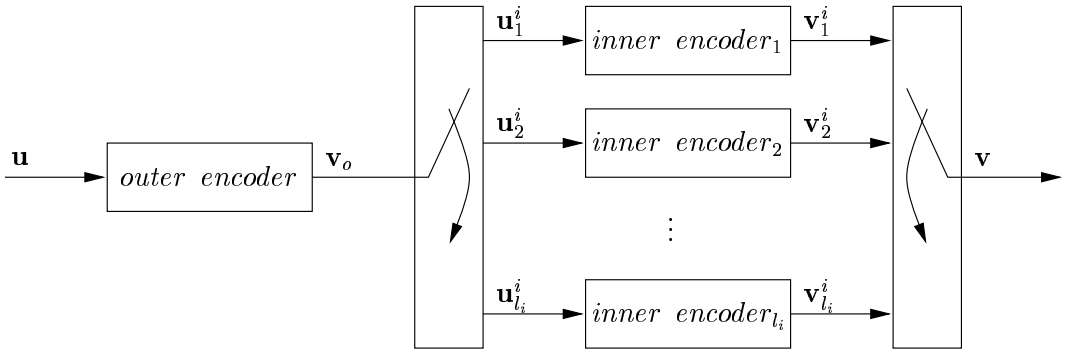



Figure 4.1: Binary woven convolutional encoder with inner warp

Again, we start our discussion by considering the encoding scheme. With binary woven convolutional codes several convolutional encoders are combined in such a way that the over-all code is also convolutional. There are two possible constructions: Woven convolutional encoders with *inner warp* as depicted in Figure 4.1 or

woven convolutional encoders with *outer warp* (see Figure 4.2). In the following discussion we will only consider the second case.

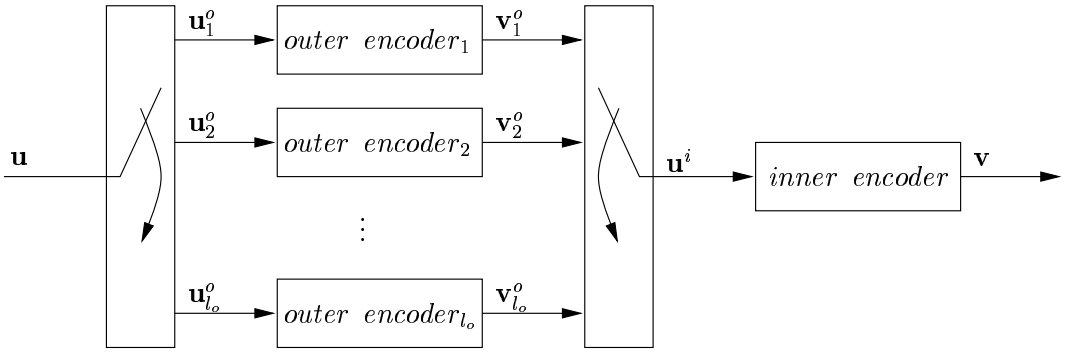



Figure 4.2: Binary woven convolutional encoder with outer warp

A binary woven convolutional encoder with outer warp consists of $l_o$ outer convolutional encoders which do not necessarily have the same generator matrices but should have the same rate $R_o = k_o/n_o$. The information sequence $\mathbf{u}$ is divided into sub-blocks of $l_o k_o$ information symbols which are fed into the parallel outer encoders. The output sequences $\mathbf{v}_1^o, \ldots, \mathbf{v}_{l_o}^o$ are written row-wise into a buffer of $l_o$ rows. These code sequences are called the *warp*. The binary code symbols are read column-wise and the resulting sequence constitutes the input sequence $\mathbf{u}^i$ of the only inner rate $R_i = k_i/n_i$ encoder, the so-called *woof* (see Figure 4.3). The over-all code is convolutional again. The woven convolutional encoder with outer warp has over-all rate $R_w = k_w/n_w$, where $k_w = l_o k_o$ and $n_w = l_o n_o n_i/k_i = l_o n_o/R_i$. Therefore, we obtain

$$R_w = R_o R_i \ . \tag{4.1}$$

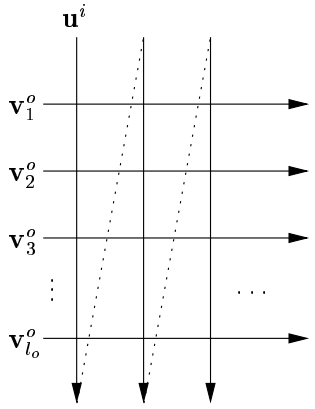



Figure 4.3: Woof and warp of a woven code

**Interleaving:** It is a well known fact that decoders for convolutional codes which work on the basis of a code trellis tend to organize decoding errors in bursts [Jor96].

Therefore interleaving is often employed in systems of concatenated convolutional codes [BM98]. We will now discuss woven codes resulting from encoder constructions with additional interleaving between outer and inner encoders.

In the following discussion we will distinguish between three different constructions presented in Figures 4.4, 4.5, and 4.6, where we assume each interleaver to independently perform a random permutation of the input symbols.

- over-all interleaving:

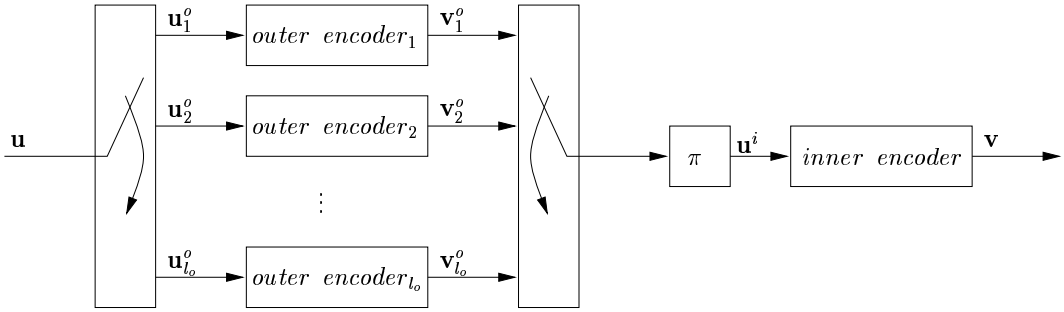



Figure 4.4: Woven encoder with over-all interleaving

- row-wise interleaving:

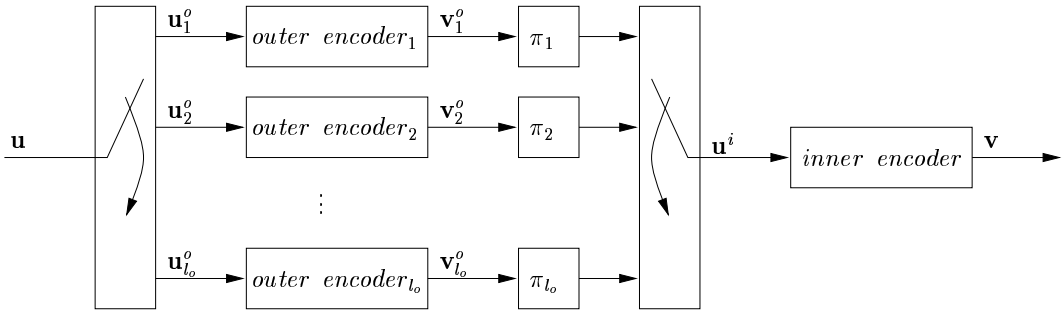



Figure 4.5: Woven encoder with row-wise interleaving

- column-wise interleaving:

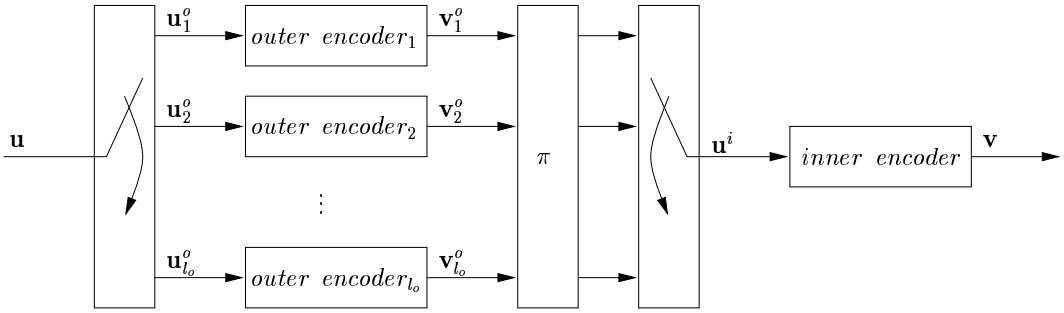



Figure 4.6: Woven encoder with column-wise interleaving

Clearly, the over-all code encoded by a woven encoder with column-wise interleaving is still convolutional, whereas it is convenient to consider the codes resulting from encoder constructions with over-all or row-wise interleaving to be block codes.

## 4.2 Decoding Scheme

In this section we will describe the iterative decoding algorithm used for the decoding of woven codes. In this discussion we again use L-values and assume symbol-by-symbol MAP decoding for the decoding of the component codes. Moreover, we presume equal probable information bits, thus $L(\mathbf{u})$ equals zero for the first decoding step. The decoding scheme for woven codes with over-all interleaving is depicted in Figure 4.7. This scheme can easily by adapted to other interleavings.

The SISO-decoder for the inner code accepts a-priori and channel L-values and delivers soft-output L-values $L(\hat{\mathbf{u}}^i)$ correspond to the outer code bits. These output symbols are de-interleaved and de-multiplexed to the inputs of the outer decoders. The outer decoders provide extrinsic values $L_e(\hat{\mathbf{v}}_j^o)$, $j \in \{1, \ldots, l_o\}$, for code bits which are used to gain the new a-priori values for the next iteration. The L-values for information bits $L(\hat{\mathbf{u}}_j^o)$ are only used in the final iteration and will then be multiplexed to the decoder output.

## 4.3 Distance Properties

### 4.3.1 Free Distance and Active Distances

Now we will discuss distance properties of the resulting woven codes. We start by considering the over-all free distance $d_{free}^w$ of woven codes encoded by binary woven convolutional encoders with outer warp.

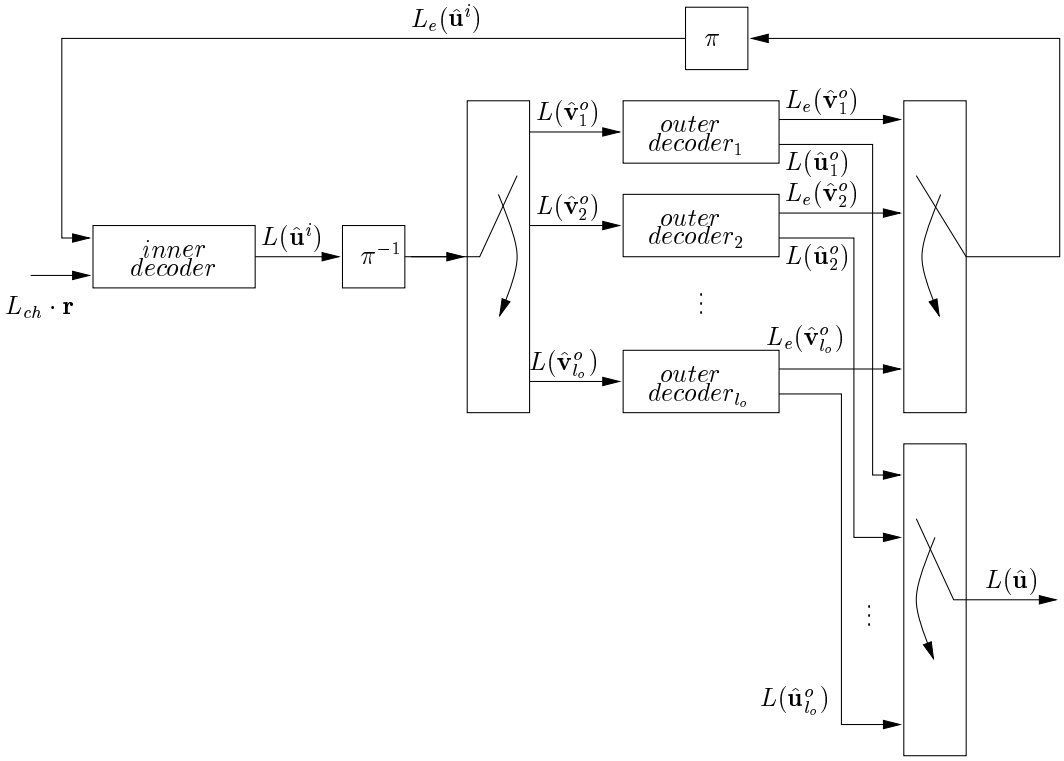



Figure 4.7: Iterative decoding scheme for woven codes

Consider the active distances of the inner encoder. Let $j_{free}^{s_i}$ denote the smallest $j$ for which

$$a_i^s(j) \geq d_{free}^i \tag{4.2}$$

holds and let $j_{free}^{r_i}$ be the smallest $j$ that satisfies

$$a_i^r(j) \geq 2d_{free}^i \quad . \tag{4.3}$$

Assume that the $l_o$ outer encoders encode the same convolutional code and that the following two conditions are fulfilled.

**Condition 1:**

$$l_o \geq (j_{free}^{s_i} + 1)k_i \tag{4.4}$$

**Condition 2:**

$$l_o \geq j_{free}^{r_i} k_i \quad . \tag{4.5}$$

Then we can lower bound the over-all free distance of the binary woven convolutional code with outer warp by

$$d_{free}^w \geq d_{free}^o d_{free}^i \quad , \tag{4.6}$$

where $d_{free}^o$ and $d_{free}^i$ are the free distances of the outer and inner code, respectively.

**Proof:** *The basic idea behind this encoder construction is that if we chose $l_o$ the number of outer encoders large enough we can guarantee that each non-zero bit in a given row will generate at least weight $d^i_{free}$ in the woven code word. But in order to proof Equation 4.6 we must additionally guarantee that there exists no other sequence $\mathbf{u}^i$ with weight $wt(\mathbf{u}^i) > d^o_{free}$ that produces an output sequence with weight smaller than $d^o_{free}d^i_{free}$.*

*If the first symbol in an information sequence to one of the outer encoders is non-zero, we assume without loss of generality that it is the first sequence $\mathbf{u}^o_1$. First, we consider the case that each non-zero bit in $\mathbf{v}^o_1$ starts a code word segment $\mathbf{v}_{[0,j]}$ of the outer code sequence which terminates within $j \leq l_o - 1$ transitions. If the first condition is fulfilled we obtain*

$$\min_j(a^r_i(j)) = d^i_{free} \leq wt(\mathbf{v}_{[0,j]}) \leq a^r_i(0) \;\; \forall j \in [0, \ldots, l_o - 1] \;\; . \tag{4.7}$$

*Next, we consider the case that the sequence $\mathbf{v}_{[0,j]}$ started by the first non-zero bit in $\mathbf{u}^o_1$ does not terminate for $j \leq l_o - 1$. Consequently, there exist symbols in $\mathbf{u}^o_1$ that do not start new sequences. Consider the segment $\mathbf{v}_{[0,l_o]}$ which terminates after $l_o$ transitions. The weight of this segment has to satisfy*

$$wt(\mathbf{v}_{[0,l_o]}) \geq a^r_i(l_o) \geq 2d^i_{free} \;\; , \tag{4.8}$$

*which is enforced by the second condition. As $a^r_i(l_o) \leq a^r_i(j) \;\; \forall j \in [l_o + 1, \ldots, 2l_o - 1]$ this is the worst case for segments which terminate in above interval.*

*We define*

$$\hat{a}^r(j) = \min_{j' \geq j}(a^r(j')) \;\; , \tag{4.9}$$

*with $\hat{a}^r(j) \geq \alpha \cdot j + \beta^r$. From $\hat{a}^r(0) = \max(d_{free}, \beta^r) = d_{free}$ it follows that*

$$\beta^r \leq d_{free} \;\; . \tag{4.10}$$

*And we obtain*

$$a^r((i-1)l_o) \geq i \cdot d_{free} \;\; , \tag{4.11}$$

*if $a^r(l_o) \geq 2d_{free}$ holds, which is enforced by the second condition. We conclude*

$$wt(\mathbf{v}_{[0,j]}) \geq i \cdot d_{free} \;\; \forall j \in [(i-1) \cdot l_o, \ldots, i \cdot l_o - 1] \;\; , \tag{4.12}$$

*and therefore Equation 4.6 holds.*

Note, that this proof holds for polynomial as well as for recursive systematic inner encoders. If we use different outer encoders with identical $d^o_{free}$, then the free distance of the resulting woven code might exceed the product of the free distances $d^o_{free}$ and $d^i_{free}$.

**Example:** (from [HJZ97]) *We use $l_0 = 16$ outer encoders with $G_i^o(D) = G_1^o(D) = (1 + D + D^2 \;\; 1 + D^2)$ and $G_{i+1}^o(D) = G_2^o(D) = (1 + D^2 \;\; 1 + D + D^2)$ alternating and $G^i(D) = (1 + D + D^2 + D^3 \;\; 1 + D^2 + D^3)$ as inner encoder. Their free distances are $d_{free}^o = 5$ and $d_{free}^i = 6$. The over-all free distance is $d_{free}^w = 35$, which exceeds the product $d_{free}^o d_{free}^i = 30$.*

**Active Distances:** We will now state lower bounds for the active distances of binary woven convolutional codes with outer warp as presented in [HJZ97].

$$\begin{aligned} a_w^r(j) &\geq \alpha_o d_{free}^i \cdot j + \beta_o^r d_{free}^i \\ a_w^c(j) &\geq \alpha_o d_{free}^i \cdot j + \beta_o^c d_{free}^i \\ a_w^{rc}(j) &\geq \alpha_o d_{free}^i \cdot j + \beta_o^{rc} d_{free}^i \\ a_w^b(j) &\geq \alpha_o d_{free}^i \cdot j + \beta_o^b d_{free}^i \\ a_w^s(j) &\geq \alpha_o d_{free}^i \cdot j + \beta_o^s d_{free}^i \end{aligned} \tag{4.13}$$

As a major result we observe that the slope $\alpha_w$ of the over-all active distances is determined by the free distance of the inner code and by the slope $\alpha_o$ of the outer codes

$$\alpha_w = \alpha_o \cdot d_{free}^i \;\; . \tag{4.14}$$

### 4.3.2 Analogy to Serial Concatenated Codes

The encoder construction depicted in Figure 4.4 with over-all interleaving can be considered as a special case of *serial concatenated convolutional codes with interleaving*. With random interleaving this construction clearly violates the conditions given in Section 4.3.1. Therefore, we cannot ensure that $d_{free}^w \geq d_{free}^o d_{free}^i$ holds. Due to the random interleaving the analysis of the distance properties of a particular woven code is a very complex task and soon becomes unfeasible with increasing code dimension. To overcome this problem Benedetto introduces a probabilistic approach to calculate the expectation of the over-all weight distribution of a serial concatenation of convolutional codes linked by a random interleaver. This approach is also applicable for woven codes with over-all random interleaving and will now be explained.

Let $A^o(I, W)$ denote the IOWEF resulting from multiplexing $l_o$ outer code words and $A^i(I, W)$ the IOWEF of the one inner encoder, respectively. We can express $A^o(I, W)$ in terms of the $l_o$ weight distributions $A_j^o(I, W)$ of the outer encoders:

$$A^o(I, W) = \prod_{j=1}^{l_o} A_j^o(I, W) \;\; . \tag{4.15}$$

With equal outer encoders and therefore $A_1^o(I, W) = \ldots = A_{l_o}^o(I, W)$ this can be simplified to:

$$A^o(I, W) = [A_1^o(I, W)]^{l_o} \;\; . \tag{4.16}$$

Assuming *uniform interleaving* as discussed in Chapter 3 we are now able to evaluate the expectation of the input-output weight enumerating function $A^w(I, W)$:

$$E[A^w(I,W)] = \sum_{l=0}^{N_o} \frac{A^o(I,l) \cdot A^i(l,W)}{\binom{N_o}{l}} \quad , \tag{4.17}$$

with $N_o = l_o(K_o + m_o)/R_o$. Like SCCC woven encoders have only one inner convolutional encoder. If we assume equal inner and outer encoders for both constructions, the only remaining difference is the number of outer encoders. Figure 4.8 exhibits the influence of $l_o$ on the IOWEF of the serialized outer encoders.

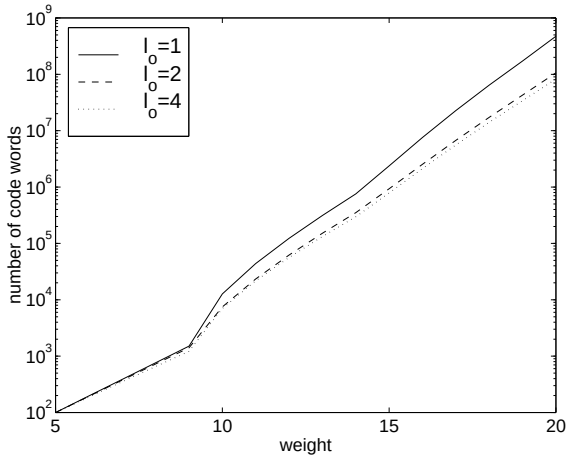



Figure 4.8: Weight distribution of $l_o$ outer memory $m_o = 2$, rate $R_o = 1/2$ encoders; $l_o \cdot K_o = 100$

## 4.4 Key Parameters

**Rate of the component codes:** The rate of the over-all woven code with outer warp is the product of the inner and outer code rate $R_w = R_o \cdot R_i$. In order to achieve moderate high over-all rates, we have to employ high rate component codes. A simple method to obtain high rate convolutional codes without increasing decoding complexity is puncturing. We will restrict ourselves to OFD rate $R = 1/2$ mother codes and rate $R = k/k+1$ punctured OFD codes. Some examples for component code rates and the resulting over-all rate can be found in Table 4.1. To evaluate these rates we have used the following rule:

$$\begin{aligned} R_w &= R_o \cdot R_i = R_1 \cdot R_2 \\ &= \frac{k_1}{k_1+1} \cdot \frac{k_1+1}{k_1+2} = \frac{k_1}{k_1+2} \quad . \end{aligned} \tag{4.18}$$

| $R$ | $1/3$ | $1/2$ | $2/3$ | $3/4$ |
|---|---|---|---|---|
| $R_1$ | $1/2$ | $2/3$ | $4/5$ | $6/7$ |
| $R_2$ | $2/3$ | $3/4$ | $5/6$ | $7/8$ |

Table 4.1: Component code rates and over-all code rate of woven codes

**Terminated codes:** If we terminate the convolutional component codes the over-all code rate decreases. Let each outer encoder encode $K_o$ symbols per transmitted block. Then each information block consists of $K_w = l_o \cdot K_o$ information bits corresponding to $N_w$ code bits, with:

$$N_w = \frac{l_o \cdot \frac{K_o + m_o k_o}{R_o} + m_i k_i}{R_i} \quad . \tag{4.19}$$

The over-all code rate is:

$$R_w = \frac{K_w}{N_w} = R_i R_o \cdot \frac{K_w}{K_w + l_o m_o k_o + R_o m_i k_i} = R_i R_o R_{loss} \quad , \tag{4.20}$$

where the term $R_{loss}$ complies the fractional rate loss of the woven code. For increasing $K_w$ this expression converges to one and becomes negligible, but for short block lengths and great numbers of outer encoders it remains significant.

**Approximating the slope of the over-all active distances:** We have already mentioned that if certain conditions are fulfilled, e.g. the number of outer encoders $l_o$ is large enough, the list $\mathcal{A}^w = (d_{free}, \alpha^w, \beta^b_w, \beta^c_w, \beta^{rc}_w, \beta^s_w)$ of a woven convolutional code with outer warp is given by the lists $\mathcal{A}^o$ and $\mathcal{A}^i$ of the inner and outer component codes. The slope $\alpha^w$ of the woven code is calculated per information tuple, i.e. for $l_o \cdot k_o$ information bits. The slope $\alpha$ of the component codes is also calculated per information tuple, which in the case of rate $R = k/k+1$ codes corresponds to one redundancy bit. In order to compare $\alpha_w$ with the slopes of OFD codes we introduce the normalized $\alpha^{norm}_w$, which is the slope per additional redundancy bit:

$$\alpha^{norm}_w = \frac{\alpha_w}{n_w - k_w} = \frac{\alpha_o \cdot d^i_{free}}{l_o \left( \frac{n_o n_i}{k_i} - k_o \right)} \tag{4.21}$$

We will now derive an approximation for $\alpha^{norm}_w$ based on the slopes $\alpha_i$ and $\alpha_o$ of the component codes. Consider the definition of $j^{r_i}_{free}$ on page 45. Let us assume that $j^{r_i}_{free}$ satisfies

$$\alpha_i j^{r_i}_{free} + \beta^{r_i} \geq 2 d^i_{free} \quad . \tag{4.22}$$

Then with Equation 4.10 and 4.5 we follow:

$$\alpha_i j^{r_i}_{free} \geq d^i_{free} \tag{4.23}$$

$$l_o \geq j^{r_i}_{free} k_i \geq \frac{k_i d^i_{free}}{\alpha^i} \quad . \tag{4.24}$$

Substituting $l_o$ in Equation 4.21 leads to:

$$\alpha_w^{norm} \leq \frac{\alpha_i \cdot \alpha_o}{n_o n_i - k_o k_i} \tag{4.25}$$

As the assumption $\alpha_i j_{free}^{r_i} + \beta^{r_i} \geq 2d_{free}^i$ does not hold for all codes, we define

$$\alpha_w^{\Pi} = \frac{\alpha_i \cdot \alpha_o}{n_o n_i - k_o k_i} \approx \alpha_w^{\Pi} \tag{4.26}$$

and use it as an approximation for the normalized slope of the woven code. In Table 4.2 we present the key parameters of woven convolutional codes with rate $R_w = 1/3$ and OFD component codes with equal memory $m_i = m_o \in \{2, \ldots, 6\}$. We observe that the approximation $\alpha_w^{norm} \approx \alpha_w^{\Pi}$ is in fact quite good.

| $m_i = m_o$ | $R_i$ | $R_o$ | $l_o$ | $d_{free}^w$ | $\alpha_w^{norm}$ | $\alpha_w^{\Pi}$ |
|---|---|---|---|---|---|---|
| 2 | 1/2 | 2/3 | 10 | 15 | 0.063 | 0.063 |
| 2 | 2/3 | 1/2 | 12 | 15 | 0.063 | 0.063 |
| 3 | 1/2 | 2/3 | 13 | 24 | 0.039 | 0.043 |
| 3 | 2/3 | 1/2 | 24 | 24 | 0.042 | 0.043 |
| 4 | 1/2 | 2/3 | 21 | 28 | 0.029 | 0.029 |
| 4 | 2/3 | 1/2 | 26 | 28 | 0.026 | 0.029 |
| 5 | 1/2 | 2/3 | 25 | 48 | 0.026 | 0.028 |
| 5 | 2/3 | 1/2 | 42 | 48 | 0.024 | 0.028 |
| 6 | 1/2 | 2/3 | 36 | 60 | 0.019 | 0.021 |
| 6 | 2/3 | 1/2 | 48 | 60 | 0.019 | 0.021 |

Table 4.2: Key parameters of woven convolutional codes with rate $R_w = 1/3$

As both parameters $\alpha_w^{norm}$ and $d_{free}^w$ are measures for the code performance we obtain two different criteria for code optimization:

- In order to enlarge the over-all free distance we have to increase the free distances of the inner and the outer codes.
- In order to achieve good over-all active distances we have to chose outer codes with great slopes $\alpha_i$ and $\alpha_o$.

Again, consider Table 4.2. We notice that with increasing memory of the component codes we gain higher over-all free distances while $\alpha_w^{norm}$ decreases. With increasing memory we also increase the fractional rate loss $R_{loss}$ which is illustrated in Figure 4.9. Moreover, we observe that for $R_i > R_o$ we usually need more outer encoders which is likely to enlarge the fraction rate loss. In Appendix A we present some tables with key parameters of rate $R_w = 1/2$ and $R_w = 2/3$ woven convolutional codes.

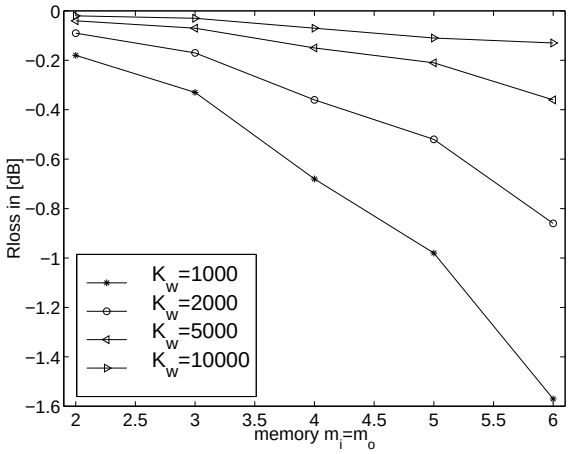



Figure 4.9: Fractional rate loss of woven codes with $R_i \cdot R_o = 1/3$

**Mapping of the inner encoder:** Up to now we have neglected the mapping of the inner encoder. The over-all code is obviously independent of different realizations of the outer generator matrices, as all equivalent encoders produce the same set of outer code sequences. The over-all code is only a subspace of the inner component code and therefore the mapping of the inner encoder becomes important.

We illustrate this fact with a simple example. Consider the serial concatenation of one outer convolutional code encoded by $\mathbf{G}^o(D)$ and one inner encoder with encoding matrix $\mathbf{G}^i(D)$. The resulting code is encoded by

$$\mathbf{G}(D) = \mathbf{G}^o(D) \cdot \mathbf{G}^i(D) \ . \tag{4.27}$$

If we select an equivalent outer encoding matrix $\mathbf{G}'^o(D) = \mathbf{T}(D)\mathbf{G}^o(D)$ the resulting generator matrix is equivalent to $\mathbf{G}(D)$.

$$\mathbf{G}'(D) = \mathbf{T}(D)\mathbf{G}^o(D)\mathbf{G}^i(D) = \mathbf{T}(D)\mathbf{G}(D) \ . \tag{4.28}$$

If we select an equivalent inner encoding matrix $\mathbf{G}'^i(D) = \mathbf{T}(D)\mathbf{G}^i(D)$ the resulting generator matrix is not equivalent to $\mathbf{G}(D)$ and we obtain a new over-all code.

$$\mathbf{G}'(D) = \mathbf{G}^o(D)\mathbf{T}(D)\mathbf{G}^i(D) \neq \mathbf{T}(D)\mathbf{G}(D) \ . \tag{4.29}$$

# Chapter 5

# Simulation Results

In this chapter we present simulation results for woven convolutional codes and woven codes with interleaving. First, we analyze the influence of key parameters such as the number of outer encoders and the memory of the component codes. Next, we examine different interleavings. As the construction of woven codes with over-all and row-wise interleaving are very similar to serial concatenated convolutional codes, we present a detailed comparison of SCCC, turbo codes and woven codes with interleaving.

## 5.1 Investigating Woven Codes

We start our discussion by investigating important parameters of woven convolutional codes without interleaving. We usually use equal inner and outer mother codes, where we employ $l_o$ outer encoders with $G_1^o(D)$ and $G_2^o(D)$ alternating. For example we may chose $G_1^o(D) = (1+D+D^2 \;\; 1+D^2)$ and $G_2^o(D) = (1+D^2 \;\; 1+D+D^2)$. We have already mentioned in the example on page 47 that this construction with alternating outer encoders may improve the distance properties of the resulting woven code.

We restrict ourselves to woven convolutional codes with outer warp employing OFD and punctured OFD convolutional component codes. The encoding matrices of the employed OFD mother codes are presented in Table 5.1, where we use systematic encoding if not mentioned otherwise. For puncturing we use deletion maps from [YKH84].

| memory | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| $\mathbf{G}(D)$ polynomial | (7 5) | (17 15) | (35 23) | (53 75) | (171 133) |
| $\mathbf{G}(D)$ systematic | (1 5/7) | (1 15/17) | (1 23/35) | (1 53/75) | (1 133/171) |

Table 5.1: Encoding matrices of OFD component codes

Despite the restriction to OFD component codes there is still a great number of free parameters allowing the design of different over-all codes:

- code dimension
- code rate
- number of outer encoders
- choice of component codes
  - choice of inner and outer code rate
  - memory of inner and outer code
  - choice of equivalent inner encoder

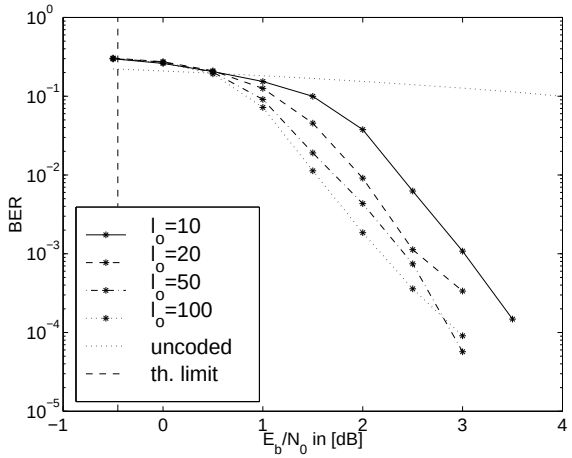
Note: As we employ sub-optimal iterative decoding and we do not know how close the results from iterative decoding are to ML decoding, we cannot conclude whether the observed results are due to code properties or to the behavior of the decoding algorithm.

### 5.1.1 Number of Outer Encoders

First of all, we consider the number of outer encoders $l_o$. The lower bounds on the free distance $d_{free}^w$ and the slope $\alpha_w$ of the active distances of woven convolutional codes are independent of the actual number of outer encoders as long as Condition 1 and 2 on page 45 are fulfilled. The normalized slope of the active distances $\alpha_w^{norm}$ decreases with a growing number of outer encoders and the fractional rate loss $R_{loss}$ increases, thus we may expect similar or even worse code performance with increasing $l_o$. On the other hand, the number of outer encoders influences the over-all distance properties although the lower bounds do not change.

We investigate the influence of $l_o$ for a rate $R = 1/3$ woven code with dimension $k = 10000$ and $m = 2$ component codes, where we employ a systematic inner encoder. In order to guarantee $d_{free}^w \geq 15$ we need at least $l_o = 10$ outer encoders. The simulation results are presented in Figure 5.1. We observe that more outer encoders lead to better bit error rates.

This result may have several reasons: First, we do not know how close the lower bounds on the free distance and the active distances are. Due to the additional interleaving it is likely that increasing $l_o$ improves the distance properties. On the other hand, the simulation results may be due to the behavior of the iterative decoding algorithm. The bit errors occurring in different outer decoders are independent and therefore increasing $l_o$ probably improves the behavior of the decoder (see [Jor96]).

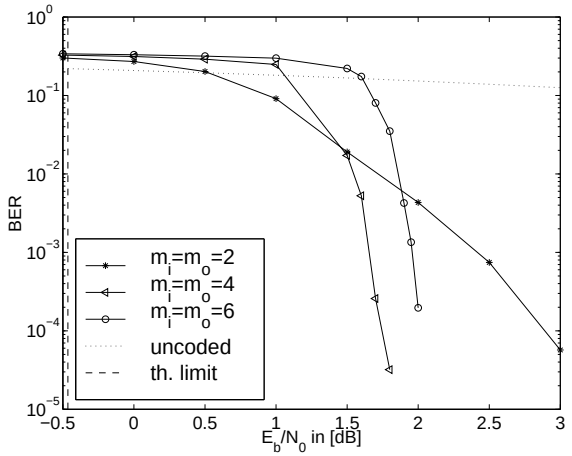



- dimension $k = 10000$
- rate $R = 1/3$
- memory $m_i = m_o = 2$
- SISO-decoding with 10 iterations

Figure 5.1: Number of outer encoders

### 5.1.2 Component Codes

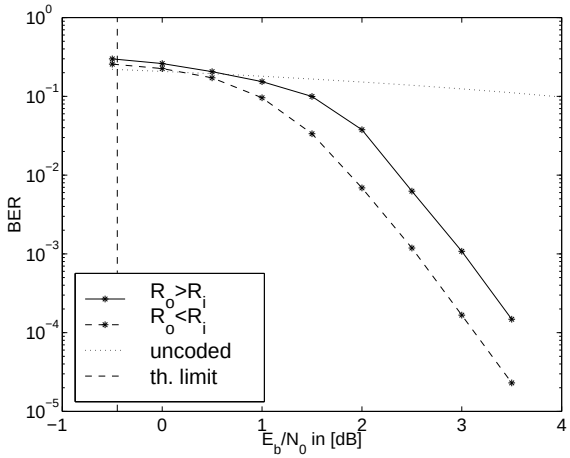
**Memory of the component codes:** Next, we consider the memory of the component codes. We have chosen equal numbers of outer encoders $l_o = 50$ for all examples depicted in Figure 5.2. Again, we investigate codes with rate $R = 1/3$ and dimension $k = 10000$, where we vary the memory of the component codes.




- dimension $k = 10000$
- rate $R = 1/3$, $R_o > R_i$
- number of outer encoders $l_o = 50$
- SISO-decoding with 10 iterations

Figure 5.2: Memory of the component codes

Here we notice that increasing memory of the component codes and therefore increasing over-all free distance does not necessarily lead to higher coding gains for bit error rates about $10^{-5}$. The woven code with $m = 4$ component codes and $d^w_{free} = 28$ performs better than the construction with $m = 6$ and $d_{free} = 60$ (see Table 4.2). But we assume that codes with high free distance perform asymptotically better than codes with low $d_{free}$.

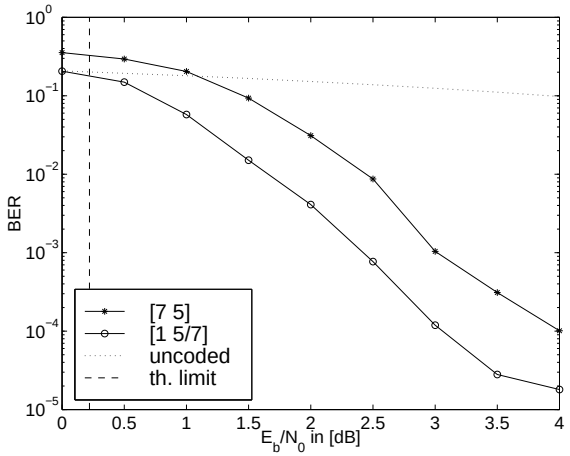



- dimension $k = 10000$
- rate $R = 1/3$
- memory $m_i = m_o = 2$
- number of outer encoders $l_o = 10$
- SISO-decoding with 10 iterations

Figure 5.3: Choice of component code rates

**Choice of component code rates:** We usually use inner and outer component codes with different code rates (see Table 4.1). If we employ punctured OFD codes with the same inner and outer mother codes the over-all code parameters $d^w_{free}$ and $\alpha^{\Pi}_w$ do not change whether we select $R_i < R_o$ or $R_i > R_o$, but we notice in Figure 5.3 that the construction with $R_i > R_o$ achieves better results.

**Mapping of the inner encoder:** If we employ two different equivalent encoders as inner encoders we obtain two different woven codes, although the two inner encoders encode the same code and the lower bounds on $d^w_{free}$ and $\alpha_w$ are the same. In Figure 5.4 we depict the simulation results for woven codes with polynomial or systematic inner encoder, respectively.




- dimension $k = 10000$
- rate $R = 1/3$
- memory $m_i = m_o = 2$
- number of outer encoders $l_o = 10$
- SISO-decoding with 10 iterations

Figure 5.4: Mapping of the inner encoder

The construction with systematic inner encoder achieves a higher coding gain. This behavior might be due to better distance properties, but may also result from the

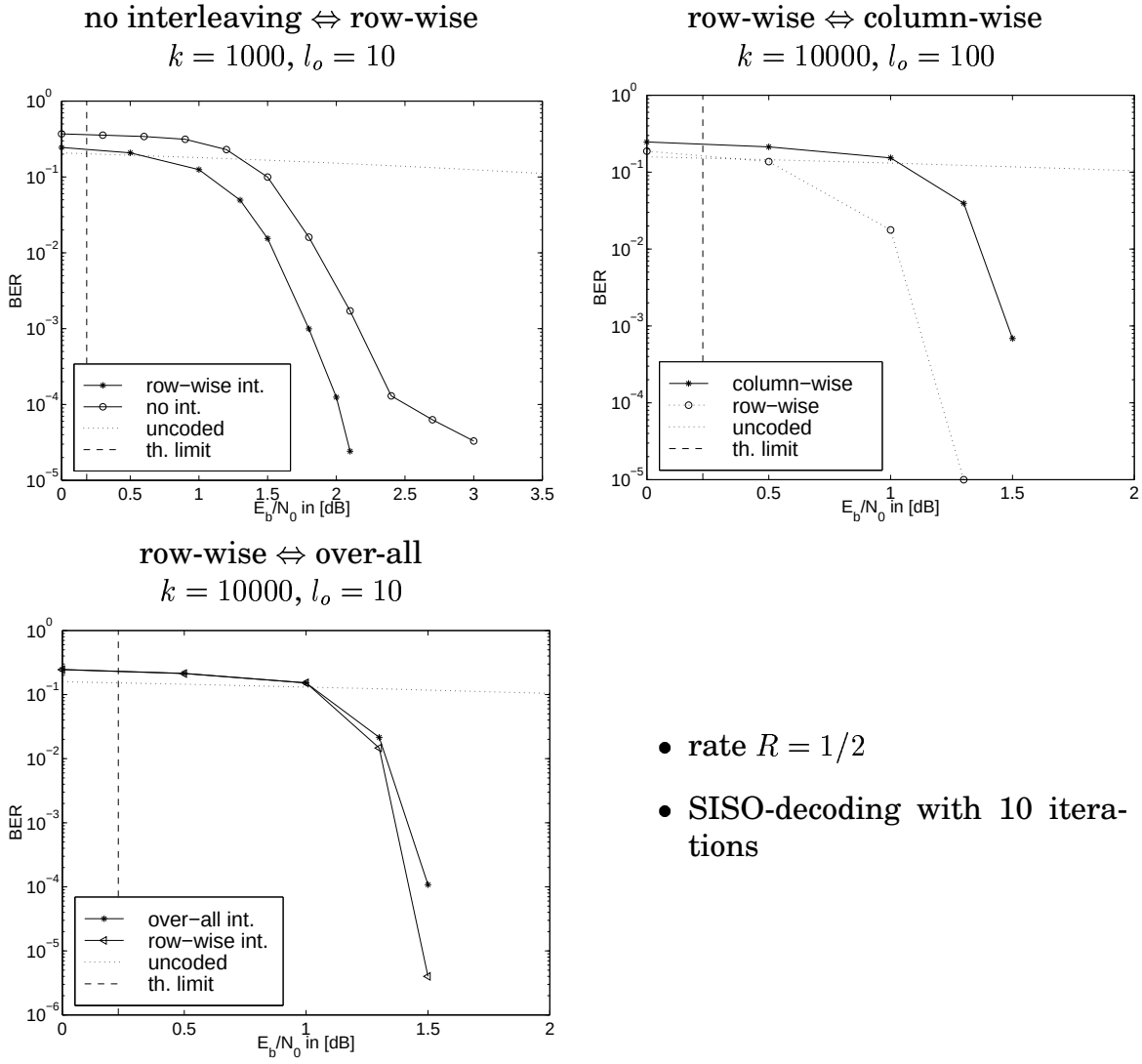
systematic encoding and therefore better bit error rates for low signal to noise ratios (see Figure 2.12).

### 5.1.3 Interleaving

Up to now we have only investigated woven convolutional codes without additional interleaving. As decoders for convolutional codes tend to organize decoding errors in bursts, interleaving is often employed between concatenated convolutional codes. We may interpret the code construction with $l_o$ outer encoders as a special kind of interleaving. In Chapter 4 we have introduced three types of additional interleaving: Row-wise, column-wise and over-all interleaving. In Figure 5.5 we compare woven convolutional codes with the three other constructions.

We notice that woven codes with row-wise interleaving achieve higher coding gains than codes with column-wise or over-all interleaving. They also perform better than woven convolutional codes without additional interleaving. For higher code dimensions the difference between woven convolutional codes and woven codes with row-wise interleaving becomes even more significant.

In Appendix B.1 we present some additional simulation results for woven codes with row-wise interleaving. We notice that systematic inner encoding should be used for woven convolutional codes as well as for woven codes with row-wise interleaving.




- rate $R = 1/2$
- SISO-decoding with 10 iterations

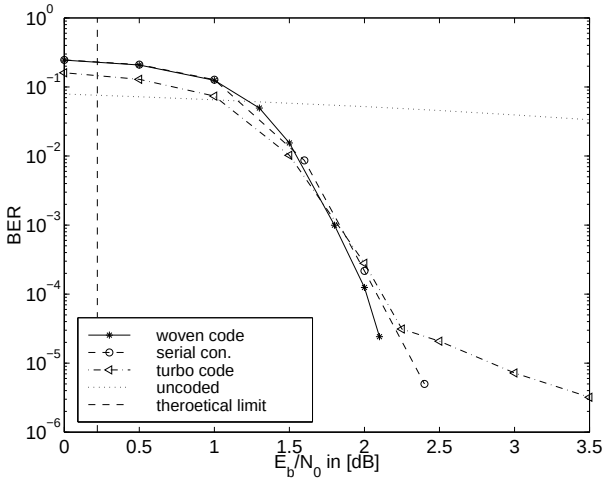
Figure 5.5: Comparison of different interleavings

## 5.2 Woven Codes in Comparison

We will now present simulation results which compare woven codes with row-wise interleaving with turbo codes and serial concatenated convolutional codes (SCCC). Although we restrict the construction for turbo codes and SCCC to encoders employing component codes with equal OFD mother codes there is still a great degree of freedom for the design of different code concatenations. The remaining parameters are:

- code dimension
- code rate
- choice of component codes
  - memory
  - generator matrix
  - puncturing

**Reference Codes:** For our simulations we take rate $R = 1/2$ codes of dimension $k = 1000$ as a reference. Based on this codes we investigate the influence of above mentioned parameters.



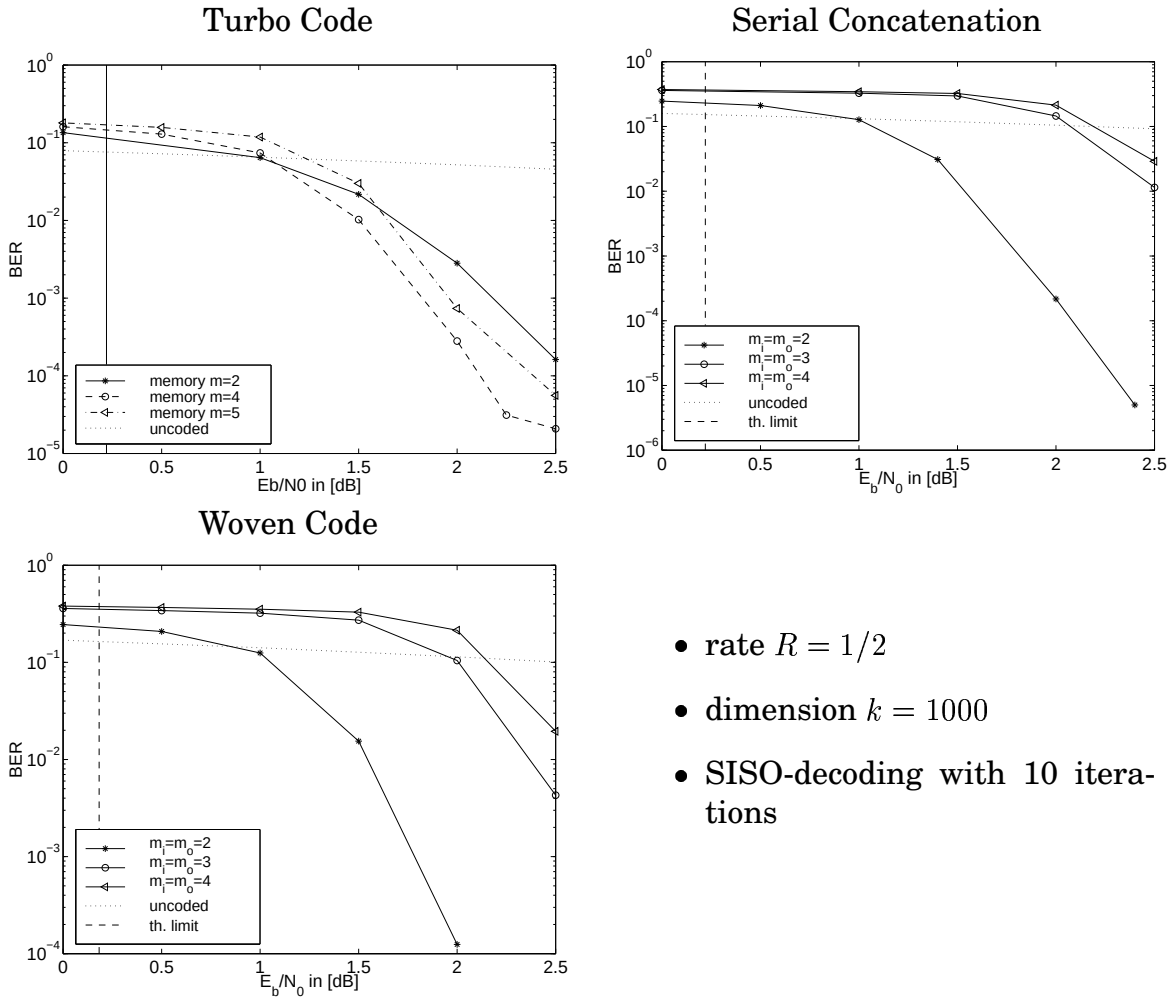

- dimension $k = 1000$
- rate $R = 1/2$
- turbo code with memory $m = 4$ $(1 \;\; 23/35)$ punctured rate $R = 2/3$ component codes; over-all random interleaving
- SCCC with memory $m = 2$ $(1 \;\; 5/7)$ punctured rate $R_i = 2/3$, $R_o = 3/4$ component codes; over-all random interleaving
- woven code with $l_o = 10$ outer and one inner memory $m = 2$ $(1 \;\; 5/7)$ punctured rate $R_i = 2/3$, $R_o = 3/4$ component codes; row-wise random interleaving
- SISO-decoding with 10 iterations

Figure 5.6: BER of the reference turbo code

All simulations consider an additive white Gaussian noise channel and BPSK modulation. The bit error rates of the three reference codes are depicted in Figure 5.6. We observe a somehow typical behavior for turbo codes. The curve differs much at low and high $E_b/N_0$. At 2.25 dB there is a 'kink'. Here we yield a BER of approximately $10^{-5}$ at an $E_b/N_0$ ratio of just 2 dB above the theoretical limit. After that the BER decreases with a much lower slope.

We know from information theory that the bit error rate for high $E_b/N_0$ ratios is determined by the minimum distance of a code [Bos98]. The progression of the BER after the 'kink' is often explained as a consequence of the small minimum distance of turbo codes [Ben96].

### 5.2.1 Memory of Component Codes




- rate $R = 1/2$
- dimension $k = 1000$
- SISO-decoding with 10 iterations

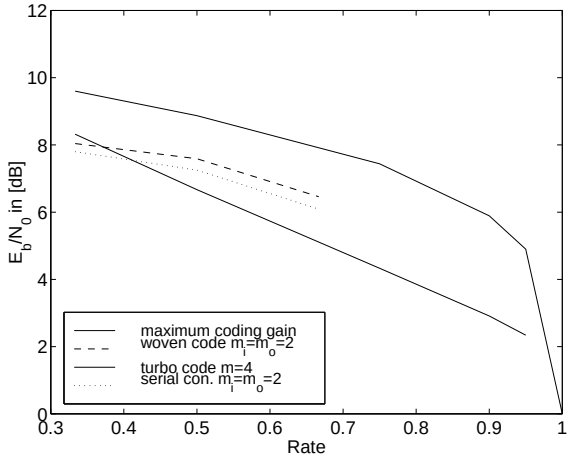
Figure 5.7: BER depending on the memory of the component codes

In Chapter 3 we investigated the expectation of the weight enumerating function of turbo codes and SCCC. These considerations have been limited to codes of small dimension. As a result we have noticed that increasing memory of the component codes may lead to more favorable weight distributions, which means we would assume better code performance presuming maximum likelihood decoding.

In Figure 5.7 we compare rate $R = 1/2$ codes with different component codes. We recognize that increasing memory does not necessarily improve the code performance for low $E_b/N_0$ ratios. With regard to near limit decoding certain component codes seem to be optimal. For turbo codes we obtain the highest coding gain with $m = 4$ component codes. For serial concatenations and woven codes $m = 2$ codes appear to be best. Both constructions achieve coding gains comparable to that of turbo codes, but with lower decoding complexity. For higher rates respectively degree of puncturing component codes with more memory may be required in order to achieve best results.

### 5.2.2 Code Rate

The Shannon limit for BPSK transmission over an AWGN channel is a function of the code rate. We can define the maximum possible coding gain for a given bit error rate as the difference between the $E_b/N_0$ ratio for uncoded BPSK and the $E_b/N_0$ ratio of the theoretical lower limit.




- rate $R = 1/2$
- dimension $k = 1000$
- SISO-decoding with 10 iterations
- BER: $10^{-5}$

Figure 5.8: Coding gain depending on the code rate

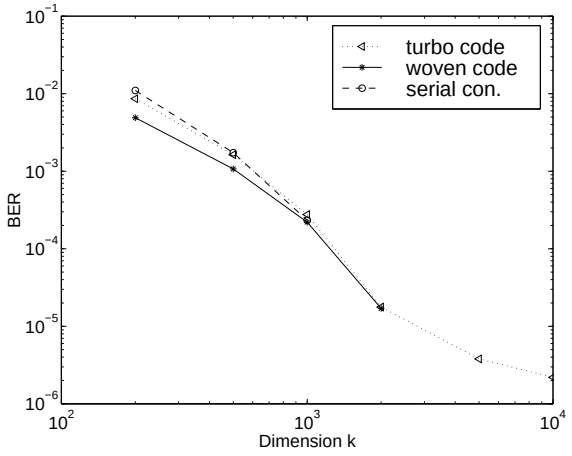
In Figure 5.8 we depict the coding gain for codes with different code rates for a arbitrarily chosen BER of $10^{-5}$. We observe that the achievable coding gain is a function of the code rate. In comparison to woven codes and SCCC, turbo codes have only superior coding gain for low rates. With serial concatenations the overall rate is the product of the component code rates, thus for high rate codes very

high rates for the component codes are required. Bit error rates depending on the code rate are presented in Appendix B.2.

We have already mentioned that in order to obtain high code rates we use punctured OFD codes. As systematic inner encoding is important, we use recursive systematic mother codes which are punctured according the optimal puncturing schemes (see for example [YKH84]). Note: The resulting encoding may be non-systematic.

### 5.2.3 Code Dimension

We know from information theory that long codes should perform better. For random codes we would expect an exponential decrease of the word and therefore also for the bit error rate. The aim of code concatenation is the construction of long codes with only a small increase in decoding complexity. The great success of turbo codes is mainly based on the fact that we are able to decode arbitrary long codes with the iterative decoding algorithm.




- rate $R = 1/2$
- SISO-decoding with 10 iterations
- $E_b/N_0 = 2db$

Figure 5.9: Coding gain depending on the code rate

With our last figure concerning code parameters we illustrate the impact of the code dimension on the code performance. In Figure 5.9 we depict the bit error rate depending on the dimension. We realize that in fact the bit error rate decreases fast with growing code dimension. But in the case of turbo codes the curve flattens for great code dimensions.

### 5.2.4 Iterative Decoding

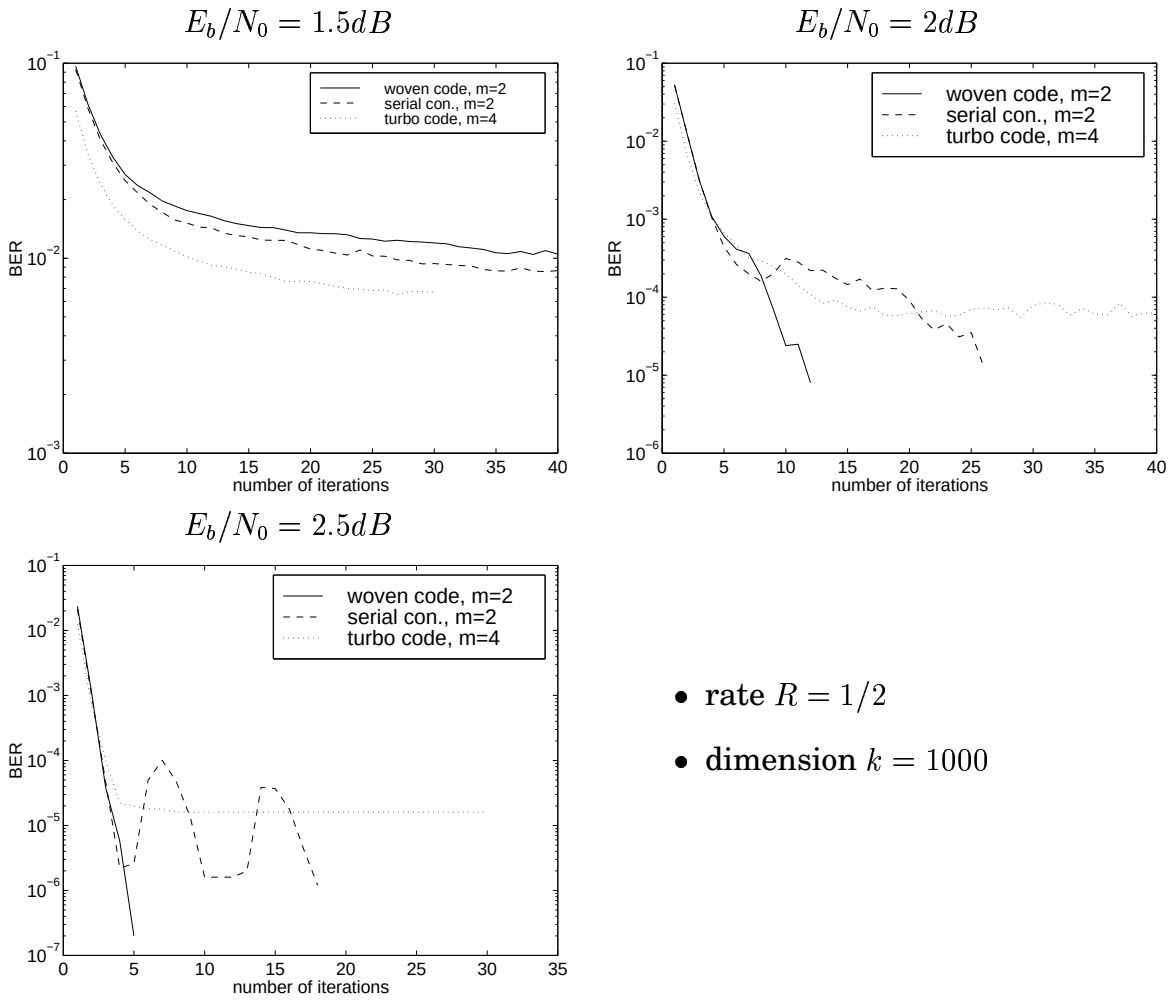
Beside the impact of code parameters the behavior of the iterative decoder influences the code performance. Concerning the iterative decoding there are a number of un-answered questions:

- What is the performance of a ML decoder?
- How close is the proposed sub-optimal iterative decoding algorithm to ML decoding?
- How sensitive is the iterative decoding algorithm to sub-optimal decoding of the component codes?
- Does the iterative algorithm converge and, if so, after what number of iterations?

Here we will only consider the last two questions. Figure 5.10 shows the BER depending one the number of iterations. Although there is no proof that the iterative decoding algorithm converges, we notice that the BER decreases fast with increasing number of iterations, where the slope depends on the $E_b/N_0$ ratio. For turbo codes the BER decreases until it approaches a certain level, also depending on the $E_b/N_0$ ratio.

In literature, we find different proposals to adjust the number of iterations to the $E_b/N_0$ ratio. But with respect to hardware implementations it seems reasonable to stop decoding after a certain number of iterations in order to achieve a constant decoding complexity. We usually stop decoding after ten iterations.

As already mentioned we use symbol-by-symbol MAP decoding to decode the component codes. In fact there are a number of different optimal and sub-optimal implementations of the algorithm from Bahl *et al.* [BCJR74]. We use the sub-optimal Max-Log-MAP algorithm. In Appendix B we present a comparison between the optimal MAP implementation using L-values and the sub-optimal Max-Log-MAP algorithm which we use throughout this chapter. Usually the achievable coding gains for optimal decoding are 0.2 - 0.5 dB higher than for sub-optimal decoding.




- rate $R = 1/2$
- dimension $k = 1000$

Figure 5.10: BER depending on the number of iterations

# Chapter 6

# Conclusions

The primal construction of woven convolutional codes with outer warp was presented in [HJZ97] by Höst *et al*. In this diploma thesis, we have investigated this new class of concatenated convolutional codes. Moreover, we have introduced new encoder constructions with additional interleaving employing terminated convolutional component codes.

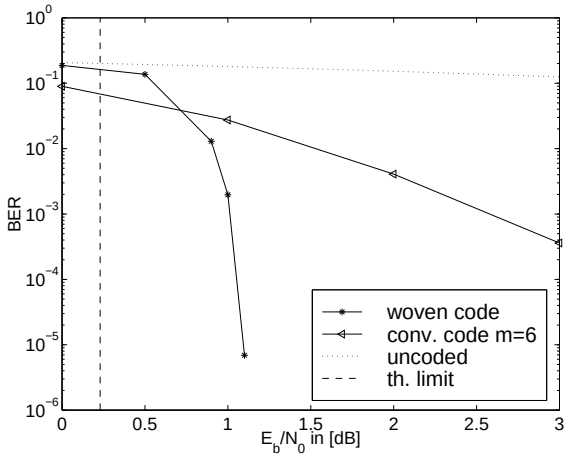
In contrast to the original introduction of woven codes, we have considered polynomial and recursive systematic inner encoders. In both cases it is proven that we can lower bound the over-all free distance $d_{free}^{w}$ of a woven convolutional code by the product of the free distances of the component codes. This conclusion is a major result of this work. In addition we have derived an approximation for the normalized slope $\alpha_{w}^{norm}$ of the active distances which indicates that the active distances of the inner and outer component codes determine $\alpha_{w}^{norm}$.

Maximum likelihood decoding of woven convolutional codes would be very complex, therefore we have suggested iterative soft-in soft-out decoding similar to the decoding of turbo codes. We have presented simulation results for woven convolutional codes as well as for woven codes with additional interleaving using this iterative algorithm. Here, we have investigated the influence of key parameters like the over-all free distance $d_{free}^{w}$ and the slope $\alpha_{w}^{norm}$. Because woven codes without additional interleaving are convolutional codes we might expect similar properties. For example, that increasing over-all free distance leads to higher coding gains at moderate high signal to noise ratios. However, the simulation results do not confirm these assumptions, which is probably due to the behavior of the iterative decoding algorithm.

Comparing the different types of interleaving, we observe that row-wise interleaving leads to the best code performance. Two major results from the simulations are that the slope $\alpha$ of the active distances of the component codes is of great importance and recursive systematic inner encoding should be used. As high slopes $\alpha$ usually correspond to convolutional codes with low memory the necessary decoding complexity remains low.

In addition to our investigations of woven codes we have discussed the encoding, decoding, and distance properties of parallel and serial concatenated codes, as all three encoder constructions are closely related. Moreover, we have presented simulation results for all three code classes. The theoretical considerations have been based on the expectations of the weight enumerating functions employing uniform interleaving. The weight enumerating function is used to upper bound the word error probability of maximum likelihood decoding. Because we use iterative decoding for all three constructions, the simulation results do not always conform the theoretical assumptions. A drawback of turbo codes is the typical 'kink' of the bit error rate, which means a small slope of the bit error rate at high signal to noise ratios. Comparing woven codes with turbo codes we have not observed the same effect.

In order to emphasize the efficiency of woven codes with row-wise interleaving, we present our final Figure 6.1. Here, we compare two rate $R = 1/2$ codes: A memory $m = 6$ convolutional code with maximum likelihood decoding and a woven code with row-wise interleaving and memory $m = 2$ component codes. By employing ten iterations of soft-in soft-out decoding for the woven code, we obtain similar calculation complexities. We notice a significantly higher coding gain for the woven code.




- dimension $k = 10000$
- rate $R = 1/2$
- memory $m_i = m_o = 2$
- SISO-decoding with 10 iterations
- optimum component decoding
- convolutional code with memory $m = 6$

Figure 6.1: Comparing woven codes with convolutional codes

This work contains the first published simulation results for woven codes with additional interleaving. A number of questions arise from these results, e.g., concerning the distance properties of the resulting codes. We have restricted ourselves to OFD and punctured OFD codes, which appear to have good active distances, but clearly further research concerning the component codes is necessary. In addition we may suggest measures which properly improve the code performance, e.g., employing tail biting in order to reduce the fractional rate loss. The choice of the particular equivalent outer encoder does not influence the over-all distance properties, thus polynomial outer encoders would enable a simple implementation of tail biting. Hopefully, we have laid the foundation for more research in this area.

# List of Abbreviations

| | |
|---|---|
| **AWGN** | additive white Gaussian noise |
| **BCJR** | Bahl, Cocke, Jelinek and Raviv |
| **BER** | bit error rate |
| **BPSK** | binary phase-shift keying |
| **BSC** | binary symmetrical channel |
| **CWEF** | conditional weight enumerating function |
| **DMC** | discrete memoryless channel |
| **IOWEF** | input-output weight enumerating function |
| **IRWEF** | input-redundancy weight enumerating function |
| **MAP** | maximum a-priori decoding |
| **ML** | maximum likelihood decoding |
| **OFD** | optimum free distance |
| **PCC** | parallel concatenated code |
| **PCCC** | parallel concatenated convolutional code |
| **SCC** | serial concatenated code |
| **SCCC** | serial concatenated convolutional code |
| **SISO** | soft-in/soft-out |
| **S/S-MAP** | symbol-by-symbol maximum a-priori decoding |
| **WEF** | weight enumerating function |

# List of Figures


2.1 Discrete memoryless channel . . . . . 4

2.2 Binary symmetric channel . . . . . 5

2.3 Channel capacity as a function of the signal to noise ratio . . . . . 7

2.4 Error weight probability $P(w)$ and the products $A_w \cdot P(w)$ dominating the word error probability . . . . . 13

2.5 Convolutional encoder with $k$ inputs and $n$ outputs . . . . . 14

2.6 Rational transfer function in controller canonical form . . . . . 14

2.7 A rate $R = 1/2$ convolutional encoder . . . . . 15

2.8 Trellis for (7 5)-convolutional code . . . . . 18

2.9 Active distances of the (7 5)-convolutional code . . . . . 20

2.10 Bit error rate of convolutional codes with different memories . . . . . 21

2.11 A single calculation step of the Viterbi algorithm . . . . . 24

2.12 S/S-MAP decoding of a systematic and non-systematic encoded codes 25

2.13 Soft-In/Soft-Out decoder and L-values . . . . . 26

3.1 Parallel Concatenated Code . . . . . 28

3.2 Serial Concatenated Code . . . . . 29

3.3 Code rate gain of a parallel concatenated code in comparison to serial concatenation . . . . . 29

3.4 Iterative decoding scheme . . . . . 30

3.5 Encoder scheme for turbo codes . . . . . 31

3.6 Decoder scheme for turbo codes . . . . . 32

3.7 Encoder scheme for serial concatenated convolutional codes with interleaving . . . . . 32

3.8 Decoding scheme for serial concatenated convolutional codes with interleaving . . . . . 33
3.9 Fully optimal interleaving . . . . . 34
3.10 Example for uniform interleaving . . . . . 35
3.11 Trellis with branch enumerators . . . . . 37
3.12 Weight distribution of a turbo code . . . . . 38
3.13 Weight distribution of turbo codes with different component codes . . 39
3.14 Weight distribution of a turbo code and an SCCC . . . . . 40

4.1 Binary woven convolutional encoder with inner warp . . . . . 41
4.2 Binary woven convolutional encoder with outer warp . . . . . 42
4.3 Woof and warp of a woven code . . . . . 42
4.4 Woven encoder with over-all interleaving . . . . . 43
4.5 Woven encoder with row-wise interleaving . . . . . 43
4.6 Woven encoder with column-wise interleaving . . . . . 44
4.7 Iterative decoding scheme for woven codes . . . . . 45
4.8 Weight distribution of $l_o$ outer memory $m_o = 2$, rate $R_o = 1/2$ encoders; $l_o \cdot K_o = 100$ . . . . . 48
4.9 Fractional rate loss of woven codes with $R_i \cdot R_o = 1/3$ . . . . . 51

5.1 Number of outer encoders . . . . . 55
5.2 Memory of the component codes . . . . . 55
5.3 Choice of component code rates . . . . . 56
5.4 Mapping of the inner encoder . . . . . 56
5.5 Comparison of different interleavings . . . . . 58
5.6 BER of the reference turbo code . . . . . 59
5.7 BER depending on the memory of the component codes . . . . . 60
5.8 Coding gain depending on the code rate . . . . . 61
5.9 Coding gain depending on the code rate . . . . . 62
5.10 BER depending on the number of iterations . . . . . 64

6.1 Comparing woven codes with convolutional codes . . . . . 66

# List of Tables


2.1 Overview of weight enumerating functions . . . . . 9

2.2 WEF family of the systematic encoded rate $R = 4/7$ Hamming code . 10

2.3 Free distances and slopes $\alpha$ for rate $R = 1/2$ OFD convolutional codes with different memories . . . . . 20

3.1 Number of code words with weight 1 to 7 . . . . . 39

4.1 Component code rates and over-all code rate of woven codes . . . . . 49

4.2 Key parameters of woven convolutional codes with rate $R_w = 1/3$ . . 50

5.1 Encoding matrices of OFD component codes . . . . . 53

A.1 Key parameters of woven convolutional codes with rate $R_w = 1/2$ . . 73

A.2 Key parameters of woven convolutional codes with rate $R_w = 2/3$ . . 73

# Appendix A

# Key Parameters of Woven Codes

| $m_i = m_o$ | $l_o$ | $d^w_{free}$ | $\alpha^{\Pi}_w$ |
|---|---|---|---|
| 2 | 12 | 9 | 0.036 |
| 3 | 24 | 16 | 0.02 |
| 4 | 26 | 20 | 0.013 |
| 5 | 42 | 42 | 0.016 |
| 6 | 48 | 48 | 0.011 |

Table A.1: Key parameters of woven convolutional codes with rate $R_w = 1/2$

| $m_i = m_o$ | $l_o$ | $d^w_{free}$ | $\alpha^{\Pi}_w$ |
|---|---|---|---|
| 2 | 16 | 4 | 0.025 |
| 3 | 28 | 9 | 0.02 |
| 4 | 40 | 12 | 0.011 |
| 5 | 72 | 20 | 0.008 |
| 6 | 104 | 25 | 0.003 |

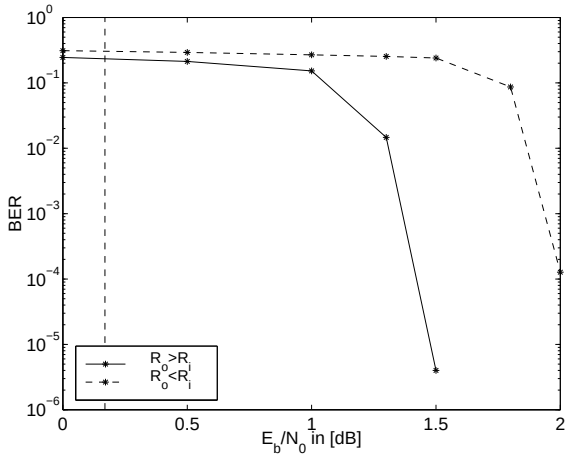
Table A.2: Key parameters of woven convolutional codes with rate $R_w = 2/3$

# Appendix B

# Additional Simulation Results

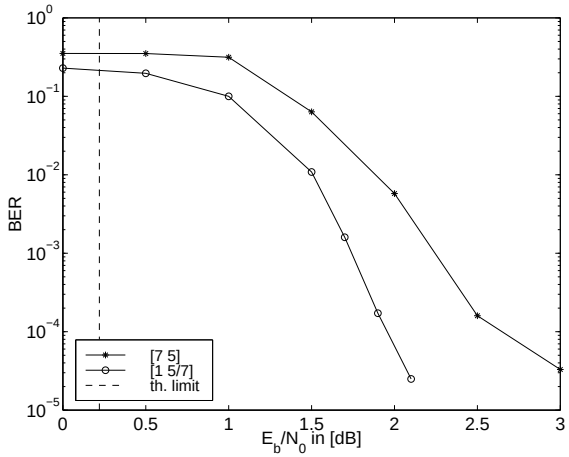
## B.1 Woven Codes with Interleaving

### Choice of component code rates




- dimension $k = 10000$
- rate $R = 1/3$
- memory $m_i = m_o = 2$
- row-wise interleaving
- number of outer encoders $l_o = 10$
- SISO-decoding with 10 iterations

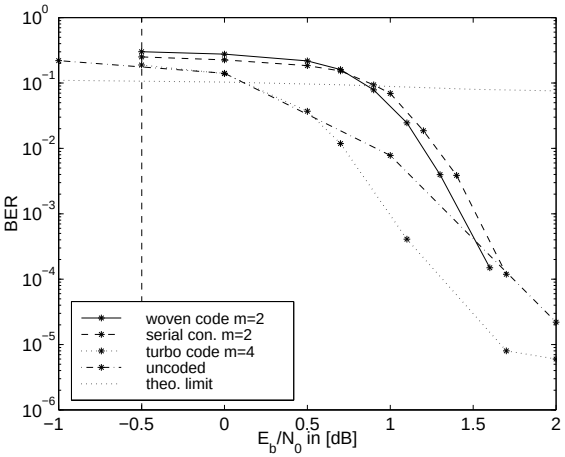
### Mapping of the inner encoder



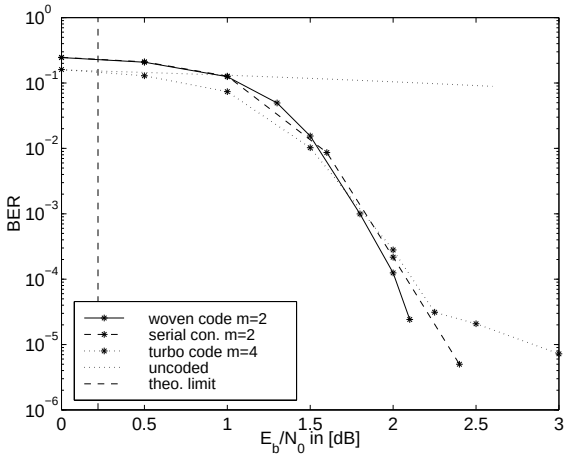

- dimension $k = 10000$
- rate $R = 1/3$
- memory $m_i = m_o = 2$
- number of outer encoders $l_o = 10$
- SISO-decoding with 10 iterations

## B.2 Code Rate

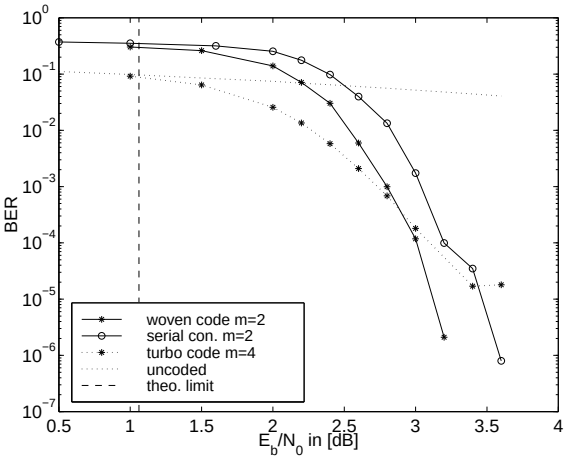
rate $R = 1/3$




rate $R = 1/2$



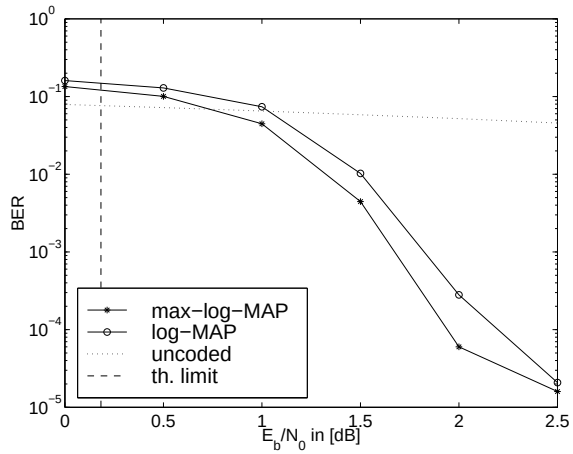

rate $R = 2/3$



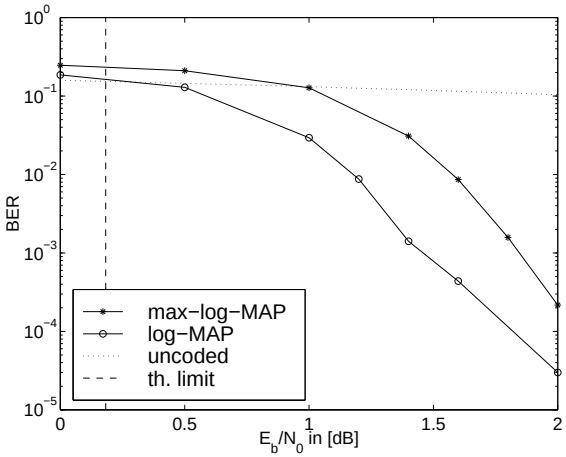

- dimension $k = 1000$
- SISO-decoding with 10 iterations

# B.3 Optimum and Sub-optimum Decoding

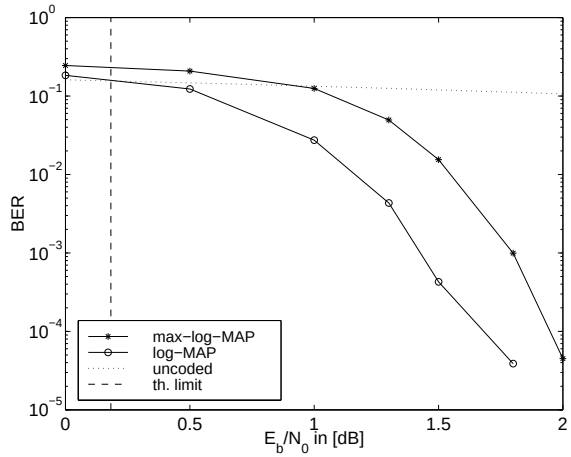
Turbo Code $m = 4$




Serial Concatenation $m = 2$




Woven Code $m = 2$,$l_o = 10$




- rate $R = 1/2$
- dimension $k = 1000$
- SISO-decoding with 10 iterations

# Bibliography


[And96] Andersen, J. D.: Turbo Codes Extended with Outer BCH Code, Electronics Letters, Vol. 32, No. 22, October 1996.

[Ben96] Benedetto, S.: Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes, IEEE Transactions on Information Theory, Vol. 42, No. 2, March 1996.

[BM96] Benedetto, S., Montorsi, G.: Design of Parallel Concatenated Convolutional Codes, IEEE Transactions on Communications, Vol. 44, May 1996.

[BM98] Benedetto, S., Montorsi, G.: Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding, IEEE Transactions on Information Theory, Vol. 44, No. 3, March 1998.

[BCJR74] Bahl, L.R.; Cocke, J.; Jelinek, F.;Raviv, J.: Optimal Decoding of Linear Codes for Minimum Symbol Error Rate, IEEE Transactions on Information Theory, Vol. IT-20, pp. 284-287, 1974.

[BGT93] Berrou, C.; Glavieux, A.; Thitimasjshima, P.: Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-codes(1), in Proc. IEEE International Conference on Communications, Geneva, Switzerland, 1993, pp. 1064-1079.

[Bos98] Bossert, M. : Kanalcodierung, B.G. Teubner Stuttgart, 1998.

[DZ97] Dahl, J.; Zyablov, V.V.: Interleaver Design for Turbo Codes; in Proc. International Symposium on Turbo Codes and Related Topics, Brest, France, 1997.

[Gal68] Gallager, R. G.: Information Theory and Reliable Communication. John Wiley & Sons, New York, 1968.

[Hag97] Hagenauer, J.: The Turbo Principle: Tutorial Introduction and State of the Art, in Proc. International Symposium on Turbo Codes, Brest, France, 1997.

[HJZ97] Höst, S.; Johannesson, R.; Zyablov, V. V.: A First Encounter with Binary Woven Convolutional Codes, in Proc. International Symposium on Communication Theory and Applications, Lake District, UK, July 13-18, 1997.

[HJZZ98] Höst, S.; Johannesson, R.; Zigangirov, K. S.; Zyablov, V. V.: Active Distances for Convolutional Codes, 1998.

[JZ96] Johannesson, R.; Zigangirov, K. S.: Convolutional Codes, to be published by IEEE Press.

[Joh92] Johannesson, R.: Informationstheorie - Grundlagen der Telekommunikation, Addison-Wesley, 1992.

[Jor96] Jordan, R.: Modellierung eines Viterbi-Decoders, Diplomarbeit, Universität Ulm, Abteilung Informationtechnik, 1996.

[Lin95] Lindner, J.: Nachrichtentechnik I,II : Manuskript zur Vorlesung Universität Ulm, 1995.

[LB98] Lucas, R.; Bossert, M.: On Iterative Soft-Decision Decoding of Linear Binary Block Codes and Product Codes, IEEE Journal on Selected Areas in Communications, Vol. 16, No. 2, 1998.

[Mas97] Massey, J. L.: Applied Digital Information Theory I, Unpublished Lecture Notes, 1997

[McE96] McEliece, R. J.: The Algebraic Theory of Convolutional Codes; Technical Report, Department of Electrical Engineering, California Institute of Technology, 1996.

[Pro95] Proakis, John G.: Digital Communications. Third Edition. New York, McGraw-Hill Book Company, 1995.

[SB98] Schnug, W.; Bossert, M.: Iterative APP-Decodierung von Faltungscodes mit Hilfe von Prüfgleichungen, Forschungsbericht, Universität Ulm, Abteilung Informationstechnik, 1998.

[Svi95] Svirid, Y. V.: Weight Distributions and Bounds for Turbo Codes, in Proc. European Transactions on Telecommunications, Vol. 6, No. 5, Sept. 1995.

[YKH84] Yasuda, Y.; Kashiki, K.; Hirata, Y.: High-Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding, IEEE Transactions on Communications, Vol. COM-32, No. 3, 1984.