

On Graph-Based Convolutional Codes

Florian Hug

Department of Electrical and Information Technology
Lund Institute of Technology

Advisor: Rolf Johannesson

March 18, 2008

Printed in Sweden
E-huset, Lund, 2009

Abstract

Today's communication systems demand increasing capacities but simultaneously decreasing error probabilities in order to provide fast and reliable data transfers. Therefore, channel coding, using proper codes with good error-correcting probabilities, like large free distances, has to be applied.

In this thesis, focusing on convolutional codes, different distance properties like the column distance, row distance, and, the most important one — the free distance — are presented. Exploiting the first two distances in the rejection rules together with the BEAST (Bidirectional Efficient Algorithm for Searching code Trees), code searches have been performed.

Existing tables of optimum free distance (OFD) convolutional codes and optimum distance profile (ODP) convolutional codes have been extended. Several new codes were found with larger free distances than previously known codes of the same rate and complexity. In particular, new tables for rate $1/2$ OFD convolutional codes of memories up to 25 and rate $1/2$ ODP convolutional codes of memories up to 40 are presented.

In the second part of this thesis, so-called woven graph codes, obtained by combining hypergraph-based codes with constituent convolutional codes are introduced. These woven graph codes are specified by their parity-check matrices and since their complexity can be rather large, an efficient method for obtaining their minimal-basic encoding matrices in minimal span form was implemented, as this form is suitable for the BEAST.

The thesis concludes by presenting promising examples of woven graph codes, with their free distance calculated by an optimized version of the BEAST. For example, in case of a rate $5/20$ woven graph code with overall constraint length 67, its free distance of 120 was verified. Such woven graph codes are of particular interest since they provide good error correcting capabilities while being iteratively decodable.

Declaration of Authorship

I certify that the thesis presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at any other University.
Lund, March 18, 2008.

Florian Hug

Acknowledgements

Now, as this work has come to its end, I want to thank all the people who supported me during the previous months.

First of all, I want to thank my supervisor Rolf Johannesson from the department of Electrical and Information Technology at Lund Institute of Technology. At all stages of this thesis, Rolf spent a lot of time discussing theory and new calculation results, and correcting my report. Moreover, his friendly attitude, understanding and open-mindedness, in scientific and personal matters, made it a pleasure to work with him.

In the same breath I have to mention Maja Lončar, Irina Bocharova, and Boris Kudryashov, who always had time for discussions and helped me to solve all upcoming problems.

Also, I want to thank all the other PhD students of the EIT department as well as the secretaries and the technical staff, especially Bertil Lindvall, who altogether made it a friendly and relaxed place to work and were always willing to listen to my problems.

A work is worthless without proper administration, especially when it has to be coordinated between two institutions. Therefore I would like to thank Martin Bossert, who organized all formalities at my home university, the University of Ulm, and made it possible for me to work on my diploma thesis in Sweden.

Special thanks to my friend and roommate Thabo Stahler, who made sure that I did not stare at the screen the whole day and never felt too good to proofread my thesis once again.

Last but not least, I want to thank my parents on whose support, of organizational and mental kind, I could always rely on.

To you, and all others, who were somehow involved in this thesis:

Thanks a lot.
Tack så mycket.
Vielen Dank.

Table of Contents

1	Introduction	1
2	Fundamentals of Channel Coding	3
2.1	General Model of a Communication System	3
2.2	Block Codes	5
2.3	Convolutional Codes	9
3	Distance and Structural Properties of Convolutional Codes	17
3.1	Column Distance	17
3.2	Row Distance	19
3.3	Code Spectrum	21
3.4	Smith Form Decomposition	21
3.5	Basic and Minimal-Basic Encoding Matrices	23
3.6	Minimal Span Form	25
4	A BEAST for Analyzing Convolutional Codes	29
4.1	Trees, Trellises, and State-Transition Diagrams	29
4.2	BEAST for Finding the Code Spectrum	33
5	Code Search	41
5.1	Exhaustive Code Search	41
5.2	Optimum Distance Profile Code Search	44
6	Woven Graph Codes	49
6.1	Graphs and Hypergraphs	49
6.2	Graph-Based Codes	51
6.3	Woven Graph Codes with Constituent Convolutional Codes	54
6.4	Encoding Matrices of Woven Graph Codes	56

7	Promising Examples of Woven Graph Codes	59
7.1	Example 1	59
7.2	Example 2	61
8	Conclusions	69
A	Software Toolbox	73
A.1	Binary Polynomials	73
A.2	Vertex Container	74
A.3	Minimal-Basic, Minimal-Span	74
A.4	Optimum Distance Profile	74
A.5	Search	75
A.6	Row Distance	75
A.7	The BEAST	76

Introduction

All digital communication systems demand fast, reliable and high-capacity data transfers over noisy wired or wireless channels. Therefore, channel coding has to be used to add redundancy in a controlled manner at the sender to help to reconstruct the transmitted sequence at the receiver. The two main classes of such channel codes are block codes and convolutional codes. This thesis focuses on convolutional codes, introduced by [Eli55].

Convolutional codes are represented either by their encoding matrix G or their parity-check matrix H and are basically characterized by:

- (i) The rate b/c , where b and c are the number of input and output symbols, respectively.
- (ii) The number of memory elements per input, used to realize the convolutional encoder in a so-called controller canonical form (CCF).
- (iii) The free distance which is the principle determiner of the code's error correction capabilities.

In the first part of this thesis a search for new convolutional codes with large free distances is presented. The distance properties, column distance, row distance, and distance profile [JZ98] are introduced in order to obtain rough estimates of the free distance. Different representations of convolutional codes in the form of state transition diagrams, code trees, and trellises are established. In order to determine the free distance efficiently, the BEAST (Bidirectional Efficient Algorithm for Searching code Trees) [BHJK01, BHJK04], can be used. Combining the distance properties and the BEAST, efficient exhaustive code searches are performed. New convolutional codes with optimum free distances were found for memories 23, 24 and 25. The search for optimum distance profile codes resulted in new optimum distance profile codes for several memories between 25 and 40.

The second part of this thesis is devoted to the so-called woven graph codes [BKJZ07, BKJZ08], which are convolutional codes constructed by combining a graph code and constituent convolutional codes of low complexity. As this code construction is based on the parity-check matrix H , structural properties of convolutional codes are exploited to obtain the corresponding generator matrix G . Introducing the Smith form decomposition, the basic, and the minimal-basic concept as well as the minimal-span form, a method to calculate an equivalent minimal-basic encoding matrix in its minimal-span form is shown. The free distance of a woven graph code, specified by such a minimal-basic encoding matrix in the minimal-span form, is calculated by an optimized version of the BEAST.

Woven graph codes have potentially very large free distances, while their structure allows iterative decoding. Promising examples of woven graph codes are presented and their free distances are determined. For example, for a rate $5/20$ woven graph code with memory 14, it will be shown that its free distance is 120.

At the end of the thesis, a short conclusion, sums up the most important results and provides an outlook on possibly interesting topics for further research.

Reader's Guideline

Chapter 2 introduces some fundamentals of channel coding and the two basic kinds of codes, namely block codes and convolutional codes. Focussing on convolutional codes, Chapter 3 presents the basic distance properties of convolutional codes, like the column distance, the row distance, the distance profile and the most important one — the free distance. Furthermore, several structural properties such as the Smith form decomposition and minimal-basic encoding matrices are covered. The representation of convolutional codes in the form of trellises and trees, together with the BEAST and its efficient implementation is described in Chapter 4. Chapter 5 introduces both the exhaustive code search and the optimum distance profile code search and presents the obtained results of several improved encoding matrices with larger free distances.

In the second part of this thesis, beginning with Chapter 6, woven graph codes are presented. As this code construction is based on the parity-check matrix H , an efficient method to obtain the corresponding generator matrix is provided. Some promising examples of woven graph codes with large free distances are highlighted in Chapter 7. Finally, this thesis is concluded by summarizing the main results in Chapter 8.

Fundamentals of Channel Coding

The *information theory* is mainly based on the idea that all communication is essentially digital and can be treated in the same way, by generating, transmitting and receiving binary digits, *bits*. The underlying *General Model of a Communication System* will be presented in Section 2.1, whereas Sections 2.2 and 2.3 introduce two basic kinds of codes, namely *block codes* and *convolutional codes*. All the results presented in the following sections are mainly based on the publications [JZ98], [Bos98], and [Lon07].

2.1 General Model of a Communication System

As already mentioned, every communication can be treated in the same way assuming a *General Model of a Communication System* as seen in Figure 2.1. Using the *separation principle* from Claude E. Shannon's landmark paper 'A Mathematical Theory of Communication' [Sha48], the *encoder* and *decoder* can be split into individual parts without loss of optimality, as illustrated in Figure 2.2.

During the (*lossless*) *source encoding*, also denoted as *data compression*, the message to be transmitted is represented by the minimum number of bits, such that the original message can be reconstructed on the receiver side without loss of information. Thereby the natural redundancy of the original message is removed. When the redundancy is removed, the bits at the output are equiprobable and identically, independently distributed. Hereinafter the information source followed by a source encoder is viewed as a equivalent binary source and its output sequence will be denoted by the information sequence \mathbf{u} .

As the propagation conditions through the channel usually introduce errors, *channel coding* is used to protect the message. The information sequence \mathbf{u} is transformed into the encoded sequence \mathbf{v} also called a *codeword* or *code sequence* by adding redundancy in a controlled manner. By exploiting this

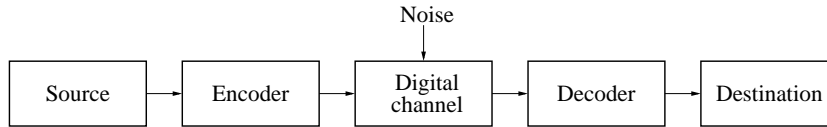


Figure 2.1: Overview of a general digital communication system.

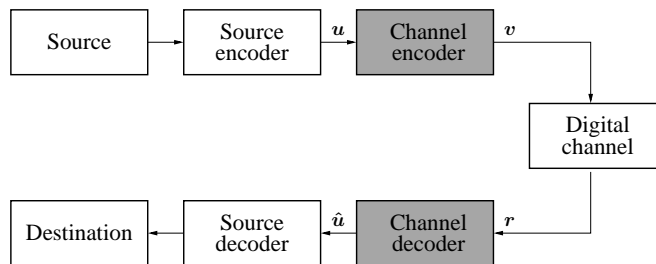


Figure 2.2: Digital communication system with separate source and channel coding. The focus on the thesis is on the grey-shaded blocks.

controlled redundancy at the receiver, most of the introduced errors can be corrected, reducing the error probability. In other words, compared to an uncoded system, a lower signaling energy is sufficient to achieve the same error probability. Consequently, by using channel coding, a cheaper and more efficient communication system can be provided.

Inside the *digital channel* the codewords \mathbf{v} are mapped onto a modulation alphabet, converted to analog waveforms, modulated, and transmitted over the (waveform) channel. Finally the received analog waveforms, modified by noise (errors) caused by the (waveform-)channel, are processed, and their (possibly quantized) values are output as a sequence \mathbf{r} of observed values.

Based on the received sequence \mathbf{r} and the knowledge of the controlled redundancy added by the channel encoder, the *channel decoder* outputs a decision $\hat{\mathbf{u}}$. If the output sequence $\hat{\mathbf{u}}$ is equal to the information sequence \mathbf{u} , the transmission was successful and all errors are corrected. Finally $\hat{\mathbf{u}}$ is forwarded to the *source decoder* which reconstructs the original information message and delivers it to its destination.

As this thesis focuses mainly on the grey-shaded blocks in Figure 2.2, the following sections will introduce the basic principles of channel coding, emphasizing on the so-called convolutional codes and their properties.

2.2 Block Codes

For simplicity, we will only deal with binary codes in the following, *i.e.*, the sequences \mathbf{u} , \mathbf{v} , and $\hat{\mathbf{u}}$ consist only of binary digits 0 and 1. Moreover, every addition is a module-2 addition, which will be denoted nevertheless by $+$ hereinafter.

In case of block codes, the entire information sequence is considered to be split up in blocks of K information bits each. These blocks, which will be called *messages*, are denoted by a row vector $\mathbf{u} = (u_0 \ u_1 \ \dots \ u_{K-1})$, where the binary digits $u_k \in \{0, 1\}$, $k = 0, \dots, K-1$. Thus we have a total set of $M = 2^K$ different messages.

An (N, K) *binary block code* \mathcal{C} of rate $R = K/N$ with $N \geq K$ is defined as the set of M distinct codewords $\mathcal{C} = \{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{M-1}\}$. Each codeword is denoted by $\mathbf{v} = (v_0 \ v_1 \ \dots \ v_{N-1})$, a row vector of length N , where $v_n \in \{0, 1\}$, $n = 0, \dots, N-1$, and corresponds to exactly one message (one-to-one mapping).

Although the number of codewords M is equal to the number of messages, the codeword lengths N is equal to or larger than the length of an individual message K . This difference $N - K \geq 0$ defines how much redundancy is added in a controlled manner. This redundancy can be used for error correction at the receiver side. In other words, while each codeword is an element of the vector space \mathbb{F}_2^N , where \mathbb{F}_2 denotes the binary (Galois) field, an (N, K) binary block code \mathcal{C} is a K -dimensional subspace of \mathbb{F}_2^N .

In case of a *linear* binary (N, K) block code, each codeword \mathbf{v} can be represented as a linear combination of K linearly independent codewords $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{K-1}$, each of length N , which form a basis of the subspace $\mathcal{C} \subset \mathbb{F}_2^N$, by

$$\mathbf{v} = \sum_{i=0}^{K-1} u_i \mathbf{g}_i \quad (2.1)$$

with $u_i \in \{0, 1\}$. This can be rewritten using matrix multiplication as

$$\mathbf{v} = \mathbf{u}G \quad (2.2)$$

where \mathbf{u} represents a message block of K information bits and G is called the $K \times N$ *generator matrix*, whose rows are the K basis codewords $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{K-1}$. From (2.2) follows immediately that the all-zero message $\mathbf{u} = \mathbf{0}$ is mapped to the all-zero codeword $\mathbf{v} = \mathbf{0}$. Moreover, for any $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, $\mathbf{x} + \mathbf{y}$ has to be in \mathcal{C} as well.

As the K linearly independent codewords can be chosen freely, the generator matrix is not unique for the underlying code \mathcal{C} . Performing elementary *row operations*, different generator matrices can be obtained, while the underlying

code \mathcal{C} remains the same. These operations preserve the set of codewords \mathbf{v} , as they are not changing the linear block code \mathcal{C} . Permuting *columns*, on the other hand, changes only the positions of code symbols inside the codeword \mathbf{v} . Thus a code \mathcal{C}' whose codewords \mathbf{v}' are permutations of the codewords of another code \mathcal{C} is said to be *equivalent* to the code \mathcal{C} . Equivalent codes have of course the same error correcting capability.

Furthermore, every code \mathcal{C} can be represented by a so-called *systematic generator matrix* G_{sys} , which performs a mapping between the message \mathbf{u} and the codeword \mathbf{v} such that the individual message symbols u_i , $i = 0, \dots, K - 1$ appear unchanged among the codeword symbols v_i , $i = 0, \dots, N - 1$.

In order to obtain a systematic generator matrix G_{sys} from any generator matrix G , the matrix G is reduced to the *reduced row-echelon form*. Thereby every row starts with a leading 1 at a distinct position, the so-called *systematic position*, which is always strictly to the right of the leading 1s of the rows above. Additionally, every column which has a leading 1 of any row contains only zeros in all other positions.

By permuting columns it is always possible to obtain a systematic generator matrix where the systematic positions are the first K positions in the codewords, *i.e.*, the generator matrix G has the form

$$G_{\text{sys}} = (I_K | P) \quad (2.3)$$

where I_K is the $K \times K$ identity matrix and P is any $K \times (N - K)$ matrix. Thus we have obtained a systematic generator matrix G_{sys} which encodes an equivalent code. The systematic codeword \mathbf{v}_{sys} can be written as

$$\mathbf{v}_{\text{sys}} = (u_0, u_1, \dots, u_{K-1}, v_K, \dots, v_{N-1}) = (\mathbf{u}, v_K, \dots, v_{N-1}). \quad (2.4)$$

Another way to specify a linear binary (N, K) block code \mathcal{C} is given by its so-called $(N - K) \times N$ *parity-check matrix* H . The rows of this matrix are linearly independent and form a subspace orthogonal to the one of the generator matrix G , that is,

$$GH^T = \mathbf{0} \quad (2.5)$$

where H^T specifies the transpose of H . From (2.2) and (2.5) follows directly that every codeword \mathbf{v} has to satisfy

$$\mathbf{v}H^T = \mathbf{0}. \quad (2.6)$$

In other words, every row of a parity-check matrix H specifies a parity-check, that has to be satisfied by the codewords of \mathcal{C} . As in the case of the generator matrix G , the parity-check matrix H is not unique as its rows can be freely chosen as long as they are linearly independent and fulfill (2.5).

In case of a systematic generator matrix G_{sys} , as it is defined in (2.3), the corresponding parity-check matrix H_{sys} is given by

$$H_{\text{sys}} = (P^T | I_{N-K}) \quad (2.7)$$

where P^T denotes the transpose of the same matrix P specified by (2.3) and I_{N-K} is the $(N-K) \times (N-K)$ identity matrix.

2.2.1 Basic Distance Properties of Block Codes

The *Hamming weight* w_{H} is defined as the number of nonzero code symbols in the codeword $\mathbf{x} \in \mathcal{C}$ of length N , *i.e.*, for a binary code,

$$w_{\text{H}}(\mathbf{x}) = \sum_{i=0}^{N-1} x_i. \quad (2.8)$$

Furthermore, the so-called *Hamming distance* d_{H} between two binary codewords \mathbf{x} and \mathbf{y} of length N , $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, is defined as the number of differing code symbols. Assuming linearity we have

$$d_{\text{H}}(\mathbf{x}, \mathbf{y}) = w_{\text{H}}(\mathbf{x} + \mathbf{y}) = \sum_{i=0}^{N-1} (x_i + y_i) \quad (2.9)$$

where x_i and y_i are the i th code symbols of \mathbf{x} and \mathbf{y} respectively.

The Hamming distance is a *metric*, since it satisfies the following properties [JZ98, p. 8]:

- (i) $d_{\text{H}}(\mathbf{x}, \mathbf{y}) \geq 0$, with equality if and only if $\mathbf{x} = \mathbf{y}$ (positive definiteness)
- (ii) $d_{\text{H}}(\mathbf{x}, \mathbf{y}) = d_{\text{H}}(\mathbf{y}, \mathbf{x})$ (symmetry)
- (iii) $d_{\text{H}}(\mathbf{x}, \mathbf{y}) \leq d_{\text{H}}(\mathbf{x}, \mathbf{z}) + d_{\text{H}}(\mathbf{z}, \mathbf{y})$ for any \mathbf{z} (triangle inequality)

Error-detecting and error-correcting capabilities depend essentially on the *minimum distance* of a code \mathcal{C} , which is given by

$$d_{\text{Hmin}} = \min_{\mathbf{x}, \mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}} \{d_{\text{H}}(\mathbf{x}, \mathbf{y})\} = \min_{\mathbf{x} \in \mathcal{C}, \mathbf{x} \neq \mathbf{0}} \{w_{\text{H}}(\mathbf{x})\} \quad (2.10)$$

where d_{H} is given by (2.9). For the binary symmetric channel (BSC), the actual codeword \mathbf{v} and the possibly erroneously received sequence \mathbf{r} are connected via the *error patterns* $\mathbf{e} = (e_0 \ e_1 \ \dots \ e_{N-1})$ by

$$\mathbf{r} = \mathbf{v} + \mathbf{e}. \quad (2.11)$$

The *number of errors* is

$$w_{\text{H}}(\mathbf{e}) = d_{\text{H}}(\mathbf{r}, \mathbf{v}) \quad (2.12)$$

and we denote the set of error patterns \mathcal{E}_t with at most t errors by

$$\mathcal{E}_t = \{\mathbf{e} \mid w_{\text{H}}(\mathbf{e}) \leq t\}. \quad (2.13)$$

A block code \mathcal{C} can correct all error patterns in \mathcal{E}_t if and only if

$$t \leq \left\lfloor \frac{d_{\text{Hmin}} - 1}{2} \right\rfloor. \quad (2.14)$$

Although, in general, it is not possible to correct all error patterns \mathcal{E}_t if (2.14) is not satisfied, these error patterns \mathcal{E}_t still can be detected as long as $t < d_{\text{Hmin}}$.

Example 2.1 (3, 2) single parity-check code

The information sequences (messages) \mathbf{u} and code sequences \mathbf{v} of a simple $R = 2/3$ linear binary block code (a so-called *single parity-check code*) with $N = 3$ and $K = 2$, i.e., $M = 2^K = 4$ distinct codewords, are listed in Table 2.1 together with the corresponding Hamming weights w_{H} . A generator matrix

\mathbf{u}	\mathbf{v}	w_{H}
00	000	0
01	011	2
10	101	2
11	110	2

Table 2.1: $R = 2/3$ linear binary block code (a so-called single parity-check code).

G for this linear binary (3, 2) block code is given by

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

while the parity-check matrix H follows as

$$H = (1 \ 1 \ 1).$$

Obviously this linear encoding rule is systematic as the first two codeword symbols v_0 and v_1 are equal to the first two message symbols u_0 and u_1 , due to the reduced row-echelon form of the encoding matrix G . The minimum distance d_{Hmin} for this code, that is the minimum weight of any nonzero codeword, is 2, thus, all error patterns with one error can be detected. (In fact, all error patterns with an odd number of errors are detectable using the single parity-check code.)

2.3 Convolutional Codes

Beside block codes, introduced in Section 2.2, the second big class of codes are the so-called *convolutional codes*, firstly introduced by [Eli55]. While block codes are based on splitting the information sequence into message blocks \mathbf{u} of length K and mapping them independently to codewords \mathbf{v} of length N , convolutional encoders continuously encode a stream of information symbols of (theoretically) infinite length.

A convolutional encoder is realized as a linear sequential circuit, consisting only of memory elements and modulo-2 adders. In this thesis we limit ourselves to convolutional encoders realized in *controller canonical form* (CCF) without feedback, *i.e.*, realized by using b shift registers, one for each input as illustrated in Figure 2.3 and Figure 2.4. Due to the memory elements, the c -tuple of code symbols appearing at the output at a given time instance t , denoted by $\mathbf{v}_t = (v_t^{(1)} v_t^{(2)} \dots v_t^{(c)})$, depends on both the corresponding input bits specified by the information b -tuple $\mathbf{u}_t = (u_t^{(1)} u_t^{(2)} \dots u_t^{(b)})$ and the previous information bits stored in the memory elements.

The input and output sequences can also be grouped according to the individual input, *i.e.*, the i th input is specified by $\mathbf{u}_t^{(i)} = (u_t^{(i)}, u_{t+1}^{(i)}, u_{t+2}^{(i)} \dots)$ and the j th output by $\mathbf{v}_t^{(j)} = (u_t^{(j)}, u_{t+1}^{(j)}, u_{t+2}^{(j)} \dots)$, respectively, where t is the discrete time instance. The convolutional code \mathcal{C} is defined as the set of all possible code sequences \mathbf{v} obtained by using all possible information sequence \mathbf{u} as inputs.

Convolutional codes are categorized by their *rate* R , their *memory* m and their *free distance* d_{free} . The rate R is the ratio between the number of bits at the input (b) and at the output (c) at any discrete time instance t , that is,

$$R = b/c. \quad (2.15)$$

If the number of memory elements used for the shift register of the i th input is ν_i , the memory m of the convolutional encoder is

$$m = \max_{1 \leq i \leq b} \{\nu_i\} \quad (2.16)$$

whereas the so-called *overall constraint length* ν is

$$\nu = \sum_{i=1}^b \nu_i. \quad (2.17)$$

In order to derive an encoding matrix G , we will consider convolutional encoding matrices without feedback as linear time invariant (LTI) systems

with b inputs and c outputs. Denoting the pulse response for every input-(i)-output-(j) combination by

$$\mathbf{g}_i^j \quad 1 \leq i \leq b \text{ and } 1 \leq j \leq c \quad (2.18)$$

the j th output $\mathbf{v}^{(j)}$ is given by the convolutional expression

$$\mathbf{v}^{(j)} = \mathbf{u}^{(1)} * \mathbf{g}_1^{(j)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(j)} + \cdots + \mathbf{u}^{(b)} * \mathbf{g}_b^{(j)} \quad (2.19)$$

$$= \sum_{i=1}^b \mathbf{u}^{(i)} * \mathbf{g}_i^{(j)}. \quad (2.20)$$

Hereinafter, the pulse responses $\mathbf{g}_i^{(j)}$ are denoted as corresponding encoding polynomials. Using a $(b \times c)$ matrix G_{conv} containing the encoder polynomials

$$G_{\text{conv}} = \begin{pmatrix} \mathbf{g}_1^{(1)} & \mathbf{g}_1^{(2)} & \cdots & \mathbf{g}_1^{(c)} \\ \mathbf{g}_2^{(1)} & \ddots & & \\ \vdots & & \ddots & \vdots \\ \mathbf{g}_b^{(1)} & \cdots & & \mathbf{g}_b^{(c)} \end{pmatrix}. \quad (2.21)$$

the input-output relation using a convolutional matrix multiplication can be expressed as

$$(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \dots, \mathbf{v}^{(c)}) = (\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \dots, \mathbf{u}^{(b)}) * G_{\text{conv}}. \quad (2.22)$$

As a matrix convolution is not easy-to-handle a better representation is given by creating separate (sub-)matrices describing the input-output relation for the individual delay instances, denoted as G_i , $0 \leq i \leq m$. Any pulse response has a length of at the most $m + 1$ or interpreted as an encoding polynomial degree m . Thereby, the number of (sub-)matrices G_i for different time instances can be upper limited by $m + 1$. In other words, all pulse responses for time instance $t = 0$ are combined to a $(b \times c)$ (sub-)matrix labeled G_0 , for time instance $t = 1$ to a (sub-)matrix G_1 , etc. until $t = m$ is reached. In case of G_0 having a full rank, which is assumed in the following, these (sub-)matrices are called *encoding (sub-)matrices* of the convolutional code \mathcal{C} with rate $R = b/c$.

Consequently the input-output relation can be simplified to

$$\mathbf{v}_t = \mathbf{u}_t G_0 + \mathbf{u}_{t-1} G_1 + \cdots + \mathbf{u}_{t-m} G_m. \quad (2.23)$$

Furthermore, it is assumed that the encoding starts at a certain time instance, say $t = 0$, and that all encoder memory elements are initially zero.

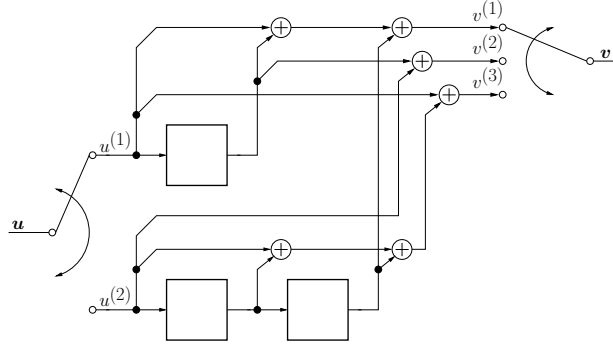


Figure 2.4: Rate $R = 2/3$ convolutional code with memory $m = 2$.

The number of memory elements of the individual inputs denoted by ν_i , as introduced in (2.16), corresponds to the highest degree of the encoding polynomials in the i th row, *i.e.*,

$$\nu_i = \max_{1 \leq j \leq c} \{ \deg(\mathbf{g}_i^{(j)}) \} \quad (2.28)$$

where $\deg(\mathbf{x})$ specifies the degree of the polynomial \mathbf{x} .

By writing (2.22) in the D -domain, the convolutional matrix multiplication is simplified to an ordinary matrix multiplication and becomes

$$\mathbf{v}(D) = \mathbf{u}(D)G(D). \quad (2.29)$$

The matrix $G(D)$ can be also expressed using the separated encoding (sub-)matrices from (2.23) by

$$G(D) = G_0 + G_1D + G_2D^2 + \dots + G_mD^m. \quad (2.30)$$

As for linear block codes, every convolutional code has many different encoding matrices which differ only in their individual mappings of information sequences \mathbf{u} to code sequences \mathbf{v} . If two different encoding matrices $G(D)$ and $G'(D)$ encode the same convolutional code \mathcal{C} they are called *equivalent* and there is a $b \times b$ rational matrix $T(D)$ of full rank such that

$$G(D) = T(D)G'(D). \quad (2.31)$$

Among all encoding matrices of an equivalent code, there always exists one, possibly rational, *systematic encoding matrix* $G_{\text{sys}}(D)$, which has the form

$$G_{\text{sys}}(D) = (I_b | R(D)) \quad (2.32)$$

where I_b denotes the $b \times b$ identity matrix and $R(D)$ is any, possibly rational, $b \times (c-b)$ matrix. The systematic encoding matrix G_{sys} defines the systematic encoding rule $\mathbf{v}_{\text{sys}}(D) = \mathbf{u}(D)G_{\text{sys}}(D)$, in which for every time instance the b -tuple of information symbols appears as part of the encoded c -tuple at the output.

Especially it has to be noted that by multiplying a systematic encoding matrix $G_{\text{sys}}(D)$ with any full rank matrices of proper size, $T(D)$, the underlying code and its properties are unchanged.

Such structural properties along with other equivalent encoding matrices, like the *basic* or the *minimal-basic encoding matrices* will be presented later on in Section 3.5.1 and Section 3.5.2.

Example 2.2 Simple rate $R = 1/2$ convolutional code

The convolutional code, whose encoder realized in CCF is illustrated in Figure 2.3, has obviously an *overall constraint length* ν and a *memory* m both equal to 2. The individual pulse responses $\mathbf{g}_i^{(j)}$ are

$$\mathbf{g}_1^{(1)} = (1, 1, 1, 0, \dots) \quad \mathbf{g}_1^{(2)} = (1, 0, 1, 0, \dots).$$

Instead of calculating the input-output relation with the convolutional multiplication as shown in (2.22) the encoded sequence can be obtained following (2.23) as

$$\mathbf{v}_t = \mathbf{u}_t G_0 + \mathbf{u}_{t-1} G_1 + \mathbf{u}_{t-2} G_2$$

with

$$G_0 = \begin{pmatrix} 1 & 1 \end{pmatrix} \quad G_1 = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad \text{and} \quad G_2 = \begin{pmatrix} 1 & 1 \end{pmatrix}.$$

Furthermore, the linear encoding rule (2.26) is given by

$$\mathbf{v} = \mathbf{u}G$$

with the encoding matrix G written as

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & \ddots \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & \ddots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Following (2.30), we finally obtain the encoding matrix $G(D)$ in the D -domain, which is given by

$$\begin{aligned} G(D) &= G_0 + G_1 D + G_2 D^2 \\ &= \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}. \end{aligned}$$

Example 2.3 Rate $R = 2/3$ convolutional code

In Example 2.2, illustrated in Figure 2.3, all *constraint lengths* ν_i , $i = 1, \dots, b$ were obviously equal to the *memory* m . Consider now the rate $R = 2/3$ convolutional encoder realized in CCF as depicted in Figure 2.4. The *overall constraint length* ν , as a sum of the *constraint length* ν_i , is given by

$$\nu = \sum_{i=1}^2 \nu_i = 1 + 2 = 3$$

whereas the *memory* m is equal to

$$m = \max_{i=1,2} \{\nu_i\} = 2.$$

For this encoding matrix only the constraint length $\nu_2 = m$, while $\nu_1 < m$. Nevertheless, the resulting equations remain the same as in Example 2.2, only the encoding polynomials and encoding matrices are different. The encoding polynomials are

$$\begin{aligned} \mathbf{g}_1^{(1)} &= 1 + D & \mathbf{g}_1^{(2)} &= D & \mathbf{g}_1^{(3)} &= 1 \\ \mathbf{g}_2^{(1)} &= D^2 & \mathbf{g}_2^{(2)} &= 1 & \mathbf{g}_2^{(3)} &= 1 + D + D^2 \end{aligned}$$

where the encoding matrices for the individual time instances are derived as

$$G_0 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad G_1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad G_2 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Moreover, the encoding matrix G used by the linear encoding rule (2.26) is given as

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & \ddots \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \ddots \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \ddots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

and, finally, by combining the individual encoding (sub-)matrixes G_i following (2.30) the encoding matrix $G(D)$ in the D -domain is

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

2.3.1 Basic Distance Properties of Convolutional Codes

The *Hamming distance* d_H as well as the *Hamming weight* w_H , already introduced for block codes in Section 2.2.1, are also valid for convolutional codes. They only differ as the individual codewords \mathbf{v} have (theoretically) infinite length, *i.e.*,

$$d_H(\mathbf{x}, \mathbf{y}) = w_H(\mathbf{x} + \mathbf{y}) = \sum_{i=0}^{\infty} (x_i + y_i) \quad (2.33)$$

where x_i and y_i are the i th code symbols of the codewords \mathbf{x} and \mathbf{y} , respectively. It has to be noted that the *metric* properties of the Hamming distance still hold in case of convolutional codes. The *free distance* d_{free} of a convolutional code \mathcal{C} is given as the minimum Hamming distance between any two different codewords

$$d_{\text{free}} = \min_{\mathbf{x}, \mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}} \{d_H(\mathbf{x}, \mathbf{y})\}. \quad (2.34)$$

Since convolutional codes are linear, (2.10) holds and (2.34) can be simplified to

$$d_{\text{free}} = \min_{\mathbf{x} \in \mathcal{C}, \mathbf{x} \neq \mathbf{0}} \{w_H(\mathbf{x})\}. \quad (2.35)$$

As error-detecting and error-correcting capabilities for block codes mostly depend on the *minimum distance* d_{min} as introduced in Section 2.2.1, the same accounts for the *free distance* d_{free} in the case of convolutional codes. Let \mathcal{E}_t be the set of error patterns with at most t errors, as defined in (2.13). A convolutional code \mathcal{C} can correct all error patterns in \mathcal{E}_t if and only if

$$t \leq \left\lfloor \frac{d_{\text{free}} - 1}{2} \right\rfloor. \quad (2.36)$$

Moreover, like for block codes, error patterns in \mathcal{E}_t with t not satisfying (2.36) still can be detected as long as $t < d_{\text{free}}$.

Distance and Structural Properties of Convolutional Codes

In this chapter several important distance measures for convolutional codes are introduced. They are useful, *e.g.*, for formulating efficient rejection rules during code search. In particular the *free distance* introduced in Section 2.3.1 will be lower and upper bounded.

Hereinafter, if not stated otherwise, a rate $R = b/c$ convolutional code \mathcal{C} with a causal information sequence

$$\mathbf{u}(D) = \mathbf{u}_0 + \mathbf{u}_1 D + \mathbf{u}_2 D^2 + \dots \quad (3.1)$$

mapped onto the code sequence

$$\mathbf{v}(D) = \mathbf{v}_0 + \mathbf{v}_1 D + \mathbf{v}_2 D^2 + \dots \quad (3.2)$$

is assumed. The mapping is performed by a linear encoding rule using the encoding matrix $G(D)$ according to

$$\mathbf{v}(D) = \mathbf{u}(D)G(D). \quad (3.3)$$

3.1 Column Distance

According to [JZ98, p. 109], the j th order *column distance* of a convolutional code \mathcal{C} , first introduced by [Cos69], is defined as the minimum Hamming distance between any two code sequences $\mathbf{v}_{[0,j]}$ resulting from the causal information sequence $\mathbf{u}_{[0,j]}$ with differing \mathbf{u}_0 . In this case $\mathbf{u}_{[0,j]}$ denotes the subsequence beginning with \mathbf{u}_0 and ending with \mathbf{u}_j , *i.e.*,

$$\mathbf{u}_{[0,j]} = (\mathbf{u}_0 \ \mathbf{u}_1 \ \dots \ \mathbf{u}_j). \quad (3.4)$$

Since convolutional codes are linear, the j th order column distance d_j^c can be expressed as the minimum Hamming weight w_H of any sequence $\mathbf{v}_{[0,j]}$ resulting from a causal information sequence $\mathbf{u}_{[0,j]}$ with $\mathbf{u}_0 \neq \mathbf{0}$, *i.e.*,

$$d_j^c = \min_{\mathbf{u}_0 \neq \mathbf{0}} \{w_H(\mathbf{v}_{[0,j]})\}. \quad (3.5)$$

By truncating the semi-infinite encoding matrix G , introduced in (2.27), after $j + 1$ columns, denoted by G_j^c ,

$$G_j^c = \begin{pmatrix} G_0 & G_1 & G_2 & \cdots & G_j \\ 0 & G_0 & G_1 & & G_{j-1} \\ 0 & 0 & G_0 & & G_{j-2} \\ & & & \ddots & \vdots \\ & & & & G_0 \end{pmatrix} \quad (3.6)$$

the sequence $\mathbf{v}_{[0,j]}$ can be expressed as

$$\mathbf{v}_{[0,j]} = \mathbf{u}_{[0,j]} G_j^c \quad (3.7)$$

and thereby (3.5) becomes

$$d_j^c = \min_{\mathbf{u}_0 \neq \mathbf{0}} \{w_H(\mathbf{u}_{[0,j]} G_j^c)\}. \quad (3.8)$$

The column distances of an encoding matrix satisfy the following conditions according to [JZ98, Thm. 3.2].

- (i) $d_j^c \leq d_{j+1}^c, j = 0, 1, 2, \dots$
- (ii) The sequence $d_0^c, d_1^c, d_2^c, \dots$ is bounded from above.
- (iii) d_j^c becomes stationary as j increases.

Furthermore, it was shown, *e.g.*, by [JZ98, Thm. 3.4] that the *free distance* d_{free} , introduced in (2.34), is equal to the upper limit of the column distances d_∞^c , *i.e.*,

$$\lim_{j \rightarrow \infty} d_j^c \stackrel{\text{def}}{=} d_\infty^c = d_{\text{free}}. \quad (3.9)$$

3.1.1 (Optimum) Distance Profile

The so-called *distance profile* of the convolutional code \mathcal{C} with the encoding matrix G of memory m is the $(m + 1)$ -tuple

$$\mathbf{d}^p = (d_0^c, d_1^c, d_2^c, \dots, d_m^c) \quad (3.10)$$

where d_j^c denotes the j th order column distance of the convolutional code \mathcal{C} .

Notice, that the distance profile is an encoder property. However, [JZ98, Thm. 3.1] showed, that the j th order column distance d_j^c is invariant over the class of *equivalent* encoding matrices, as long as G_0 has full rank, which is fulfilled per definition.

In order to introduce the distance profile of a code we introduce the so-called minimal-basic encoding matrices, which will be discussed in more detail in Section 3.5.2. Currently it is sufficient to know that every minimal-basic encoding matrix describes a realization of the convolutional code in CCF with the fewest memory elements possible. Moreover, the set of row-wise highest degrees ν_i of two equivalent minimal-basic matrices is the same up to a rearrangement [JZ98, Thm. 2.26]. Thus, considering only minimal-basic encoding matrices, we have a unique value of the memory m .

Consequently the *distance profile of a convolutional code*, encoded by an minimal-basic encoding matrix $G_{mb}(D)$ of memory m , is defined as the $(m+1)$ -tuple

$$\mathbf{d}^p = (d_0^c, d_1^c, d_2^c, \dots, d_m^c) \quad (3.11)$$

where d_j^c denotes the j th order column distance of G_{mb} and is the same for all equivalent minimal-basic encoding matrices.

Following [JZ98, p. 112], a distance profile \mathbf{d}^p is said to be superior to another distance profile $\mathbf{d}^{p'}$, of same rate R and memory m , if there is an l such that

$$d_j^c \begin{cases} = d_j^{c'} & j = 0, 1, \dots, l-1 \\ > d_j^{c'} & j = l. \end{cases} \quad (3.12)$$

A convolutional code \mathcal{C} has an *optimum distance profile* (ODP), if there is no other code \mathcal{C}' , of the same rate R and memory m , having a better distance profile.

In the following a property especially useful for the ODP code search presented in Section 5.2, will be introduced. Given an encoding matrix $G'(D)$ with each of its polynomials $g(D)$ having at most degree m' , $G'(D)|_m$ denotes the truncation of every polynomial $g(D)$ to degree $m \leq m'$. Thereby it follows, that the two encoding matrices $G(D)$ with memory m and $G''(D) = T(D)G'(D)|_m$, where $T(D)$ is any $b \times b$ non-singular matrix, are equivalent over the first memory length m . Consequently they also share the same distance profile \mathbf{d}^p .

3.2 Row Distance

As a counterpart of the *column distance* in Section 3.1, the so-called *row distance* according to [Cos69] will be introduced.

First of all the *zero state driving information sequence* \mathbf{u}^{ZS} has to be defined as the information sequence which causes all memory elements to be successively filled with zeros. For a rate $R = b/c$ convolutional code with memory m and the length of this sequence is at most bm and lasts for m time instances. Therefore the sequence will be denoted by $\mathbf{u}_{[t,t+m]}^{ZS}$ starting at time $t + 1$. Obviously the length of the zero state driving information sequence can be less than bm , *e.g.*, if there are already zeros stored in the first parts of the memory elements. However, to simplify notation, $\mathbf{u}_{[t,t+m]}^{ZS}$ will be used for these cases as well.

The j th order *row distance* d_j^r can now be defined as the minimum Hamming weight w_H of the code sequence $\mathbf{v}_{[0,j+m]}$ resulting from the causal information sequence $\mathbf{u}_{[0,j+m]} = (\mathbf{u}_{[0,j]}\mathbf{u}_{[j+1,j+m]}^{ZS})$ with $\mathbf{u}_{[0,j]} \neq \mathbf{0}$. In other words, the j th order row distance is the minimum Hamming weight of any code sequence that results from any nonzero information sequence of j b -tuples followed by the zero state driving information sequence as stated above.

Similarly to (3.6), by truncating the semi-infinite encoding matrix G from (2.27) after $j + 1$ rows, denoted by G_j^r ,

$$G_j^r = \begin{pmatrix} G_0 & G_1 & \dots & G_m & & & \\ & G_0 & G_1 & \dots & G_m & & \\ & & G_0 & G_1 & \dots & G_m & \\ & & & \ddots & & & \ddots \\ & & & & G_0 & G_1 & G_m \end{pmatrix} \quad (3.13)$$

the j th order row distance can be written as

$$d_j^r = \min_{\mathbf{u}_{[0,j]} \neq \mathbf{0}} \{w_H(\mathbf{u}_{[0,j]}G_j^r)\}. \quad (3.14)$$

The truncation of the encoding matrix G after $j + 1$ rows as introduced in (3.13) is known as *zero-tail termination (ZT)*. After having mapped the information sequence $\mathbf{u}_{[0,j]}$ to the appropriate code sequence, the ZT forces every state to be driven to the all-zero state. Notice that the *row distance* is an encoding matrix property, not a code property.

According to [JZ98, Thm. 3.5] the row distances satisfy the following conditions (similarly to the conditions of the column distances)

- (i) $d_{j+1}^r \leq d_j^r, j = 0, 1, 2, \dots$
- (ii) $d_j^r > 0, j = 0, 1, 2, \dots$
- (iii) d_j^r becomes stationary as j increases.

Thus, defining the lower limit of the row distance as d_∞^r , the following equation is obtained

$$0 < \lim_{j \rightarrow \infty} d_j^r \stackrel{\text{def}}{=} d_\infty^r \leq \dots \leq d_2^r \leq d_1^r \leq d_0^r. \quad (3.15)$$

Obviously, as the column distance d_i^c is the minimum Hamming weight of a path of length $i + 1$, diverging from the zero state in the first step, it follows that

$$d_i^c \leq d_j^r \quad \forall i, j. \quad (3.16)$$

Moreover, by combining the conditions of column and row distance, equation (3.9) and [JZ98, Thm. 3.6], we obtain

$$d_0^c \leq d_1^c \leq \dots \leq d_\infty^c = d_{\text{free}} \leq d_\infty^r \leq \dots \leq d_1^r \leq d_0^r. \quad (3.17)$$

Notice that, the free distance d_{free} serves as one of the main properties in characterizing and evaluating convolutional codes. By using (3.17), this distance can easily be lower and upper bounded by the column and row distances, respectively.

3.3 Code Spectrum

As already mentioned, the free distance d_{free} of a rate $R = b/c$ convolutional code \mathcal{C} , with memory m , defined by (2.34), will be used as the main criterion in evaluating convolutional codes. However, it is often more convenient to have more information about the code's distance structure. Let $n_{d_{\text{free}}+i}$, $i = 0, 1, 2, \dots$, denote the number of code sequences \mathbf{v} with Hamming weight $d_{\text{free}} + i$, $i = 0, 1, 2, \dots$. Then $n_{d_{\text{free}}+i}$, $i = 0, 1, 2, \dots$, is called the $(i + 1)$ th *spectral component*. Moreover, the sequence

$$n_{d_{\text{free}}+i}, i = 0, 1, 2, \dots \quad (3.18)$$

is known as the *code spectrum* for the given convolutional code \mathcal{C} and serves as a secondary parameter in evaluating convolutional codes regarding their distance structure. Notice that the spectrum is a code property, not an encoding matrix property.

3.4 Smith Form Decomposition

A very useful decomposition for polynomial matrices is given by the *Smith form decomposition*. This decomposition plays a major role when discussing the structural properties of convolutional encoding matrices later on.

Let $G(D)$ be any $b \times c$ binary polynomial matrix with $b \leq c$ and rank r . Applying the *Smith Form Decomposition* on $G(D)$, this matrix can be decomposed as

$$G(D) = A(D)\Gamma(D)B(D) \quad (3.19)$$

To obtain a *minimal-basic encoding matrix* $G_{\text{mb}}(D)$ we have to perform the following steps, starting with a *basic encoder matrix* $G(D)$ [For70]:

- (i) If $[G(D)]_h$ has full rank, $G(D)$ is a *minimal-basic encoding matrix* according to the theorem stated above and we can stop. Otherwise, we continue with the next step.
- (ii) Find a set of rows $[\mathbf{r}_{i_1}], [\mathbf{r}_{i_2}], \dots, [\mathbf{r}_{i_d}]$ of $[G(D)]_h$, such that $\nu_{i_j} \geq \nu_{i_k}$, $k < j$ and

$$[\mathbf{r}_{i_1}] + [\mathbf{r}_{i_2}] + \dots + [\mathbf{r}_{i_d}] = \mathbf{0}. \quad (3.23)$$

Denote the corresponding rows in $G(D)$ by $\mathbf{r}_{i_1}, \mathbf{r}_{i_2}, \dots, \mathbf{r}_{i_d}$. Then add

$$D^{\nu_{i_d} - \nu_{i_1}} \mathbf{r}_{i_1} + D^{\nu_{i_d} - \nu_{i_2}} \mathbf{r}_{i_2} + \dots + D^{\nu_{i_d} - \nu_{i_{d-1}}} \mathbf{r}_{i_{d-1}} \quad (3.24)$$

to the i_d th row of $G(D)$. Finally call this new matrix $G(D)$ and start from the beginning.

Hereinafter, if not stated otherwise, all encoding matrices $G(D)$ are assumed to be *minimal-basic* encoding matrices. Thereby unnecessary complexity can be avoided without any essential loss of generality.

3.6 Minimal Span Form

For linear block codes, a so-called *minimal trellis construction* [For88] minimizes the number of vertices at each depth in the *code trellis* (cf. Section 4.1).

For convolutional codes it is possible to obtain a *minimal trellis* in a similar way, using a *minimal-basic* encoding matrix $G_{\text{mb}}(D)$ in its so-called *minimal span form*, which will be introduced as follows. Notice that using the BEAST (cf. Section 4.2), the lowest calculation complexity is only obtained in case of using encoding matrices in their equivalent minimal span form.

Let us assume a minimal-basic encoding matrix $G_{\text{mb}}(D)$ for a rate $R = b/c$ convolutional code \mathcal{C} and denote each row by \mathbf{r}_i , $i = 1, 2, \dots, b$. For each row \mathbf{r}_i , the leftmost column containing the polynomial with the shortest delay s_i in the i th row will be denoted by $\text{start}(\mathbf{r}_i)$. Similarly, $\text{end}(\mathbf{r}_i)$ denotes the rightmost column containing the polynomial of the highest degree h_i in the i th row. The value $\text{start}(\mathbf{r}_i)$ will be denoted as the *start* of i th row, whereas $\text{end}(\mathbf{r}_i)$ specifies the *end* of the i th row.

A minimal-basic encoding matrix $G_{\text{mb}}(D)$ is said to be in its minimal span form, if there is not more than one row starting (start) or ending (end) at the same column, that is, for any $i, j = 1, 2, \dots, b$, $i \neq j$

$$\text{start}(\mathbf{r}_i) \neq \text{start}(\mathbf{r}_j) \quad (3.25)$$

$$\text{end}(\mathbf{r}_i) \neq \text{end}(\mathbf{r}_j). \quad (3.26)$$

Hereinafter, such a minimal-basic encoding matrix in minimal span form will be denoted by $G_{\text{mbms}}(D)$.

For a given minimal-basic encoding matrix $G_{\text{mb}}(D)$ for a convolutional code \mathcal{C} , the *minimal span form* can be obtained by performing the following steps. As we are considering only minimal-basic encoding matrices, the corresponding polynomials $g_{i,j}(D)$ are delay-free, thus, that the shortest delay of every row \mathbf{r}_i is $s_i = 0$.

- (i) Reduce the encoding matrix $G_{\text{mb}}(D)$ to its polynomial row-echelon form. Thereby every row \mathbf{r}_i starts with polynomial with shortest delay $s_i = 0$ at a distinct position, which is always strictly to the right of the polynomials with shortest delay 0 in the rows above. Thereby the starting positions $\text{start}(\mathbf{r}_i)$ for all rows $i = 1, 2, \dots, b$ are at distinct position and (3.25) is fulfilled.
- (ii) In order to also fulfill (3.26), we check every row \mathbf{r}_i , $i = b, (b-1), \dots, 1$, not yet marked as ‘done’. If there is no row \mathbf{r}_j , $j = i-1, \dots, 1$ ending at the same column, *i.e.*, $\text{end}(\mathbf{r}_i) \neq \text{end}(\mathbf{r}_j)$, $j = i-1, \dots, 1$, the row \mathbf{r}_i fulfills (3.26) and is marked as ‘done’. If all rows \mathbf{r}_i , $i = b, (b-1), \dots, 1$ are marked as done, we have successfully determined the minimal span form.

Otherwise, denote the corresponding rows \mathbf{r}_j , $j = i-1, \dots, 1$, ending at the same column as the i th row \mathbf{r}_i by $\mathbf{r}_{k_1}, \mathbf{r}_{k_2}, \dots, \mathbf{r}_{k_l} \in \mathcal{K}$, with $|\mathcal{K}| = l \leq (i-1)$.

For every row $\mathbf{r}_{k_m} \in \mathcal{K}$, $m = 1, \dots, l$, we denote its highest degree by h_{k_m} , while the highest degree in row \mathbf{r}_i is given by h_i . Depending on these highest degrees, we distinguish between two possibilities:

- If $h_i \leq h_{k_m}$, we add row \mathbf{r}_i multiplied by $D^{(h_{k_m}-h_i)}$ to row \mathbf{r}_{k_m} . Additionally we mark the row \mathbf{r}_i as ‘done’. If this causes the (new) highest degree h_{k_m} of row \mathbf{r}_{k_m} to increase, all ‘done’-flags are removed and we start again from the beginning of step (ii).
- However, if $h_i > h_{k_m}$, row \mathbf{r}_{k_m} will be multiplied by $D^{(h_i-h_{k_m})}$ and added to row \mathbf{r}_i . In this case row \mathbf{r}_{k_m} is marked as ‘done’. Similarly, if thereby the (new) highest degree h_i of row \mathbf{r}_{k_m} increases, all ‘done’-flags are removed and we start again from the beginning of step (ii).

Example 3.1 Smith Form decomposition and minimal-basic encoding matrix in minimal span form

Given the rate $R = 2/3$ convolutional encoding matrix with memory $m = 2$ from Example 2.3, we apply the Smith form decomposition and obtain an

equivalent minimal-basic encoding matrix in minimal span form. Although the original encoding matrix given by

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & D & 1+D+D^2 \end{pmatrix} \quad (3.27)$$

already fulfills the minimal-basic property ($[G(D)]_h$ has full rank), by applying the previously mentioned steps, we obtain another equivalent minimal-basic encoding matrix in minimal span form.

In a first step, using the Smith form decomposition, the encoding matrix $G(D)$ can be written as the product of the diagonal matrix $\Gamma(D)$ and the elementary matrices, mentioned in Section 3.4, namely $P_{i,j}$ and $R_{i,j}(p(D))$.

$$\begin{aligned} G(D) &= \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \times \\ &\quad \begin{matrix} R_{2,1}(1+D+D^2) & \Gamma(D) \end{matrix} \\ &\times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1+D^2+D^3 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \times \\ &\quad \begin{matrix} R_{2,3}(1+D^2+D^3) & R_{3,2}(1) & P_{1,3} \end{matrix} \\ &\times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1+D & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & D & 1 \end{pmatrix} \\ &\quad \begin{matrix} R_{3,1}(1+D) & R_{3,2}(D) \end{matrix} \end{aligned}$$

Finally the pre- and post-multipliers $A(D)$ and $B(D)$ are obtained as the products of $P_{i,j}$ and $R_{i,j}(p(D))$.

$$\begin{aligned} G(D) &= \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \times \\ &\quad \begin{matrix} A(D) & \Gamma(D) \end{matrix} \\ &\times \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & D^2+D^3 & 0 \\ 1 & 1 & 0 \end{pmatrix} \\ &\quad \begin{matrix} B(D) \end{matrix} \end{aligned}$$

Following Section 3.5.1, an equivalent basic encoding matrix $G_b(D)$ is given by the first $b = 2$ rows of $B(D)$, that is

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

Although, $G_b(D)$ is an equivalent basic encoding matrix, the minimal-basic property is not yet fulfilled, as $[G_b(D)]_h$ has not full rank.

$$[G_b(D)]_h = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Following the proposed algorithm stated above, an equivalent minimal-basic encoding matrix can be easily obtained. Therefore, denote by $[\mathbf{r}_{i_1}]$ and $[\mathbf{r}_{i_2}]$ the first and the second row of $[G_b(D)]_h$, respectively, with $\nu_{i_2} > \nu_{i_1}$. The set of rows fulfilling (3.23) is consequently given by

$$[\mathbf{r}_{i_1}] + [\mathbf{r}_{i_2}] = \mathbf{0}.$$

Following (3.24), the first row \mathbf{r}_{i_1} multiplied by $D^{\nu_{i_2} - \nu_{i_1}} = D^2$ has to be added to the second row \mathbf{r}_{i_2} , leading to

$$G_{\text{mb}}(D) = \begin{pmatrix} 1+D & D & 1 \\ 1 & D^2 & D^2 \end{pmatrix} \quad (3.28)$$

and

$$[G_{\text{mb}}(D)]_h = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

As $[G_{\text{mb}}(D)]_h$ has full rank, we can stop with the algorithm as we have obtained an equivalent minimal-basic encoding matrix $G_{\text{mb}}(D)$.

Notice that the encoding matrices (3.27) and (3.28) are equivalent. Moreover both matrices are minimal-basic and have the same set of row-wise highest degrees ν_i .

As a last step the minimal-basic encoding matrix should be transformed to its equivalent *minimal span form*. Following Section 3.6 the leftmost column with the shortest delay (**start**) and the rightmost column with the highest degree (**end**) for every row are calculated as

$$\begin{aligned} \text{start}(\mathbf{r}_1) &= 1 & \text{end}(\mathbf{r}_1) &= 2 \\ \text{start}(\mathbf{r}_2) &= 1 & \text{end}(\mathbf{r}_2) &= 3. \end{aligned}$$

Obvious, as $\text{start}(\mathbf{r}_1) = \text{start}(\mathbf{r}_2)$ we have not yet obtained the minimal span form. According to the algorithm introduced in Section 3.6 the encoding matrix $G_{\text{mb}}(D)$ has to be reduced to its polynomial row-echelon form, to obtain different starting positions, which follows as

$$G_{\text{mbms}}(D) \begin{pmatrix} 1+D & D & 1 \\ D & D+D^2 & 1+D^2 \end{pmatrix}$$

Thereby we have obtained an equivalent minimal-basic encoding matrix $G_{\text{mbms}}(D)$ (3.1) in its minimal span form, as every row has a distinct starting (**start**) and a distinct ending positions (**end**).

A BEAST for Analyzing Convolutional Codes

Every convolutional code \mathcal{C} can be represented by a graph, *e.g.*, a *code tree*, *code trellis*, or *state-transition diagram*. Such graphs are structures consisting of points, also called *nodes* or *vertices*, and lines connecting them, called *branches* or *edges*. They are quite useful for analyzing code properties and for decoding, and will be introduced in Section 4.1.

Based on these graphs, a lot of different algorithms for finding the *free distance*, the first *spectral components* or decoding received sequences \mathbf{r} have been developed. The well-known *Viterbi algorithm* [Vit67] is, *e.g.*, based on the representation of the convolutional code in the form of a code trellis. Nowadays, the most efficient algorithm for finding the *code spectrum* (*cf.* Section 3.3) and decoding convolutional codes, the *BEAST*, introduced by [BHJK01] and [BHJK04], will be discussed in Section 4.2.

4.1 Trees, Trellises, and State-Transition Diagrams

The *state* (also called *encoder state*) of a system encoding a convolutional code \mathcal{C} , at time t , specifies the past history and is, together with the current and future input symbols $\mathbf{u}_i^{(j)}$, $i \geq t$, sufficient to determine its current and future output symbols $\mathbf{v}_i^{(j)}$, $i \geq t$. In case of a convolutional encoder, each state is defined by the contents of the individual memory elements. According to the overall constraint length, introduced in (2.17), which specifies the total number of memory elements in the CCF realization, and due to the fact that each memory element can store either a 0 or a 1, there are in total 2^ν different states. For a convolutional encoder, realized in CCF, the states at time t are

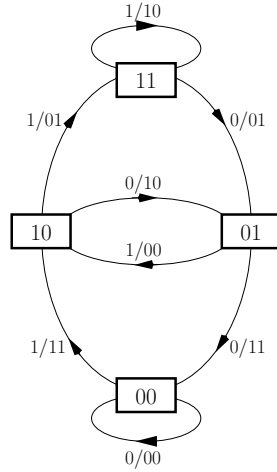


Figure 4.1: State-diagram of a rate $R = 1/2$ convolutional encoder $G(D) = (1 + D^2 \quad 1 + D + D^2)$ with memory $m = 2$.

determined directly by the past input symbols $\mathbf{u}_i^{(j)}$, $i < t$ according to

$$\boldsymbol{\sigma}_t = \left(u_{t-1}^{(1)} \ u_{t-2}^{(1)} \ \dots \ u_{t-\nu_1}^{(1)} \ u_{t-1}^{(2)} \ u_{t-2}^{(2)} \ \dots \ u_{t-\nu_2}^{(2)} \ \dots \ u_{t-1}^{(b)} \ u_{t-2}^{(b)} \ \dots \ u_{t-\nu_b}^{(b)} \right). \quad (4.1)$$

In case of a rate $R = 1/c$ convolutional encoder with memory m realized in CCF without feedback, the encoder states are simplify

$$\boldsymbol{\sigma}_t = (u_{t-1} \ u_{t-2} \ \dots \ u_{t-m}). \quad (4.2)$$

State-Transition Diagram

The so-called *state-transition diagram* for a rate $R = 1/2$ convolutional encoder given in Figure 2.3 with memory $m = 2$ is illustrated in Figure 4.1, where each node is labeled by one of $2^\nu = 2^m = 4$ encoder states. The branches between any two nodes $\boldsymbol{\sigma}_t$ and $\boldsymbol{\sigma}'_t$ represent possible state transitions $\boldsymbol{\sigma} \mapsto \boldsymbol{\sigma}'$ and are labeled by the corresponding input/output tuples \mathbf{u}/\mathbf{v} . In general, a state-transition diagram consists of 2^ν nodes, where 2^b branches leave and arrive to each node. For example, starting with the all-zero state $\boldsymbol{\sigma} = (00)$ at time $t = 0$ and entering the information sequence $\mathbf{u} = (1 \ 1 \ 0 \ 0 \ \dots)$ the encoded sequence will be $\mathbf{v} = (11 \ 01 \ 01 \ 11 \ \dots)$ by visiting the *encoder states* in the order $(00) \mapsto (10) \mapsto (11) \mapsto (01) \mapsto (00)$.

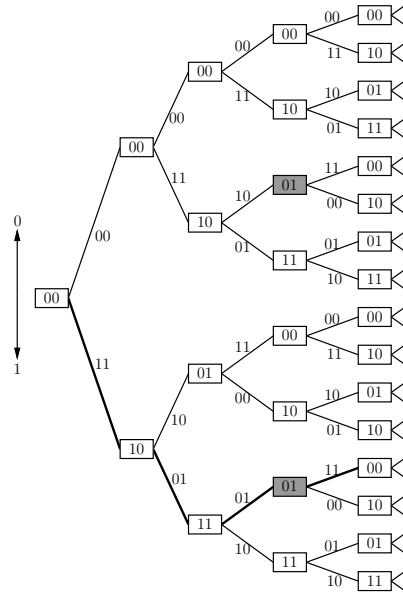


Figure 4.2: Code tree of a rate $R = 1/2$ convolutional encoder $G(D) = (1 + D^2 \quad 1 + D + D^2)$ with memory $m = 2$.

Code Trees

A rate $R = b/c$ convolutional code \mathcal{C} with memory m can be also represented by a so-called *code tree* as illustrated for a rate $R = 1/2$ convolutional code \mathcal{C} in Figure 4.2. The leftmost node is called *root* and is assigned the depth $d = 0$.

As this is a $b = 1$ convolutional code, two branches (in general 2^b) leave each node, representing the two possible input symbols, and lead to nodes at the next depth. For rate $R = 1/c$ codes, a branch leaving upwards corresponds to the input digit $u_t = 0$ whereas a branch leaving downwards corresponds to the input digit $u_t = 1$. Each node is labeled by its corresponding encoder state σ . The branches between two nodes at different depths are labeled by a c -tuple of code digits, the corresponding output symbols $\mathbf{v} = (v_t^{(1)} \ v_t^{(2)} \ \dots \ v_t^{(c)})$. In case of $b > 1$, *i.e.*, several input symbols entering at time instance t , every branch has to be labeled additionally by the corresponding input sequence \mathbf{u} . Clearly, the input sequence $\mathbf{u} = (1 \ 1 \ 0 \ 0 \ \dots)$ will be encoded as $\mathbf{v} = (11 \ 01 \ 01 \ 11 \ \dots)$ by simply following the branches in the code tree (the bold path in Figure 4.2).

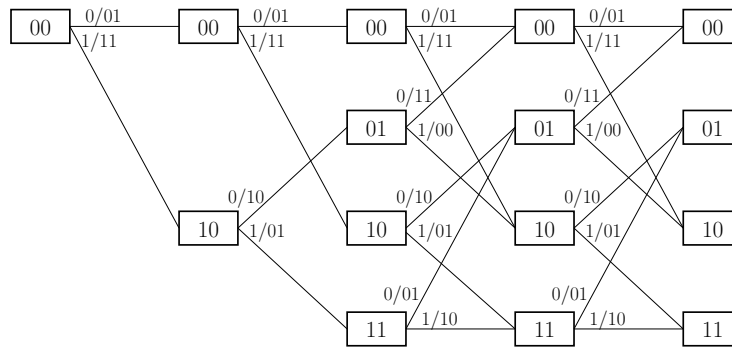


Figure 4.3: Code trellis of a rate $R = 1/2$ convolutional encoder $G(D) = (1 + D^2 \quad 1 + D + D^2)$ with memory $m = 2$.

Code Trellis

Studying the code tree in Figure 4.2 carefully, we notice that at each depth, the number of nodes doubles. Additionally, at any depth $d \geq 3$, more than one node is labeled by the same encoder state, *e.g.*, the two gray marked nodes with encoder state $\sigma = (01)$ at depth $d = 3$. Notice that in Figure 4.2 the actual encoder state only depends on the last two input symbols (in general on the last m input b -tuples).

Merging all nodes with the same encoder state σ and depth d , the size of the code tree can be reduced and we obtain the so-called *code trellis*, as illustrated in Figure 4.3. The code trellis has 2^v nodes at each depth (except the first m stages).

Starting from the all-zero state (the *root*) with depth $d = 0$, it takes m steps to populate the whole encoder memory with information symbols, which is referred as the *start-up phase*. Afterwards the so called *steady-state* is reached. There, the trellis has a *time-invariant* structure, *i.e.*, the structure of the branches leaving and entering the nodes at each depth remains the same (*cf.* Figure 4.3; steady-state starts with depth $d = 2$).

As for the state-transition diagram, every of the 2^v nodes in the *code trellis* is labeled with an *encoder state* σ , whereas the branches leaving σ_d at depth d and arriving at σ_{d+1} at the next depth $d + 1$, are labeled by the corresponding input/output tuples \mathbf{u}/\mathbf{v} for the state transition $\sigma_d \mapsto \sigma_{d+1}$.

4.2 BEAST for Finding the Code Spectrum

Instead of calculating the code spectrum by the so-called *path weight enumerator* function as presented in [JZ98, p. 114], this section introduces an alternative algorithm: *BEAST* — Bidirectional Efficient Algorithm for Searching code Trees. The BEAST was introduced by [BHJK01] and [BHJK04] as an efficient way to find the weight spectrum of both convolutional and block codes. Furthermore, with some variations, the BEAST is also suitable for maximum-likelihood decoding, *i.e.*, finding the best matching codeword (*cf.* [Lon07, Ch. 3]).

As we are particularly interested in finding the code spectrum, *i.e.*, in case of convolutional codes the free distance d_{free} and the first spectral components $n_{d_{\text{free}}+i}$, $i = 0, 1, 2, \dots$, we will focus on the original version of the BEAST.

A rate $R = b/c$ convolutional code \mathcal{C} with memory m , consists of the set of codewords $\mathcal{C} = \{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots\}$. Every nonzero codeword \mathbf{v} can be represented by a detour, *i.e.*, a path through the *code trellis* starting at the all-zero root node ξ_{root} and, in case of a noncatastrophic encoding matrix, finding its way back to the all-zero state, the toor node ξ_{toor} . Notice that the BEAST actually operates on the corresponding *code tree*, obtained by the *code trellis* by skipping the merging of vertices with equal states and depth.

Before starting with the description of the BEAST, some basic notations have to be introduced. A node in the code tree will be denoted by ξ and has a unique parent node ξ^P and 2^b children, referred to as ξ^C . Furthermore, every node has two characterizing parameters, its state $\sigma(\xi)$ and its weight $w(\xi)$. As the BEAST is a bidirectional algorithm we will have two code trees, a *forward code tree* \mathcal{F} starting at ξ_{root} and a *backward code tree* \mathcal{B} starting at the end of the tree, the ξ_{toor} .

Assume we want to find the number of codewords \mathbf{v} of weight w and any length N , which corresponds to paths from ξ_{root} to ξ_{toor} . Then, for every codeword \mathbf{v} there exists an intermediate node $\sigma(\xi) \neq \mathbf{0}$, such that

$$w_{\mathcal{F}}(\xi) = \left\lfloor \frac{w}{2} \right\rfloor + j, \quad w_{\mathcal{B}}(\xi) = \left\lceil \frac{w}{2} \right\rceil - j, \quad j = 0, 1, \dots, c-1 \quad (4.3)$$

where $w_{\mathcal{F}}(\xi)$ is the accumulated Hamming weight of the code sequence segment $\xi_{\text{root}} \mapsto \xi$ and $w_{\mathcal{B}}(\xi)$ the accumulated Hamming weight of the second part of the code sequence $\xi \mapsto \xi_{\text{toor}}$. The additional term $j = 0, 1, \dots, c-1$ is necessary as each branch is labeled by a c -tuple and therefore can contribute a weight of at most c . In the following the *forward weight* $\lfloor w/2 \rfloor$ will be replaced by f_w whereas the *backward weight* $\lceil w/2 \rceil$ will be replaced by b_w . Both weights can be chosen freely as long as they fulfill

$$f_w + b_w = w. \quad (4.4)$$

According to (4.3), we can split the search into two individual and independent searches. A *forward search*, searching for all paths $\xi_{\text{root}} \mapsto \xi$ with a weight of $f_w + j$ and a *backward search* for all paths $\xi \mapsto \xi_{\text{toor}}$ with weight $b_w - j$, $j = 0, 1, \dots, c - 1$.

In order to find the number of codewords efficiently for a given weight w , the forward weight f_w and the backward weight b_w should be balanced roughly, such that the number of vertices stored in the forward and backward set, are roughly equal.

Notice that, in case of an extremely asymmetric distribution like, *i.e.*, $f_w = w - 1$ and $b_w = 1$ for a rate $R = 1/2$ convolutional code \mathcal{C} , the c forward sets would contain almost the whole code trellis, while only the toor will be stored in one of the c backward sets. Thereby we loose most of the efficiency of the BEAST, which is based on the bidirectional search for codewords.

Having defined a certain suitable distribution of f_w and b_w , the BEAST finds the number of codewords with weight $w = f_w + b_w$ of a convolutional code \mathcal{C} as follows:

- (i) *Forward search*: Starting at the root node ξ_{root} , extend the *forward code tree* to obtain c sets indexed by $j = 0, 1, \dots, c - 1$ containing only the states $\sigma(\xi)$ of all nodes ξ satisfying

$$\mathcal{F}_{+j} = \{ \xi \mid w_{\mathcal{F}}(\xi) = f_w + j, w_{\mathcal{F}}(\xi^{\text{P}}) < f_w, \sigma(\xi) \neq \mathbf{0} \}. \quad (4.5)$$

- (ii) *Backward search*: Starting at the toor ξ_{toor} , extend the *backward code tree* to obtain c sets indexed by $j = 0, 1, \dots, c - 1$ containing only the states $\sigma(\xi)$ of all nodes ξ satisfying

$$\mathcal{B}_{-j} = \{ \xi \mid w_{\mathcal{B}}(\xi) = b_w - j, w_{\mathcal{B}}(\xi^{\text{C}}) > b_w, \sigma(\xi) \neq \mathbf{0} \}. \quad (4.6)$$

- (iii) *Matching*: For every pair $\{\mathcal{F}_{+j}, \mathcal{B}_{-j}\}$, $j = 0, 1, \dots, c - 1$, count the number of matching node pairs $\{\xi, \xi'\}$ with equal states, *i.e.*, $\sigma(\xi) = \sigma(\xi')$, $\xi \in \mathcal{F}_{+j}$ and $\xi' \in \mathcal{B}_{-j}$. Consequently, the number of codewords with weight w follows as

$$n_w = \sum_{j=0}^{c-1} \sum_{(\xi, \xi') \in \mathcal{F}_{+j} \times \mathcal{B}_{-j}} \chi(\xi, \xi') \quad (4.7)$$

where χ is the match-indicator function defined as

$$\chi(\xi, \xi') = \begin{cases} 1, & \text{if } \sigma(\xi) = \sigma(\xi') \\ 0, & \text{else.} \end{cases} \quad (4.8)$$

4.2.1 Implementation Aspects

In order to achieve an efficient practical implementation of the BEAST, several aspects should be observed:

- As the memory usage grows with the sizes of the forward and backward code trees, for some convolutional codes \mathcal{C} it will exhaust the available memory. In such a situation, the vertices should be directly written into files instead of being stored in the memory. The individual implementation of the BEAST has to be modified, to perform all operations (like sorting, matching, etc.) directly inside files on a hard disc.
- Instead of matching directly every node ξ of the forward set with every node ξ' of the backward set, it is much more efficient to sort each set before matching. This leads to the following algorithm:
 - (i) *Sorting*: Sort the forward and the backward sets according to their states in the same descending lexicographical order. By using proper algorithms this can be accomplished with a complexity of

$$|\mathcal{F}_{+j}| \log(|\mathcal{F}_{+j}|) + |\mathcal{B}_{-j}| \log(|\mathcal{B}_{-j}|) \quad (4.9)$$

where $|\cdot|$ denotes the number of elements in the set.

- (ii) *Matching*: Search for vertices with matching codewords in \mathcal{F}_{+j} and \mathcal{B}_{-j} , $j = 0, 1, \dots, c-1$, by using the following algorithm for every pair for sets. Denote the current active node in the forward set \mathcal{F}_{+j} by F_s and the corresponding active node in the backward set \mathcal{B}_{-j} by B_t . Initialize both of them with the first entry of their set, *i.e.*, $F_{s=1}$ and $B_{t=1}$. Moreover, we need a counter *count* = 0 for counting the number of matching codewords and a reference pointer *ref* = 0 to store the beginning of ‘clusters’ of vertices with the same states in the backward set. Notice that these ‘clusters’ may contain only one vertex.

As long as neither the end of the forward set nor the end of the backward set is reached, *i.e.*, $s \leq |\mathcal{F}_{+j}|$ and $t \leq |\mathcal{B}_{-j}|$, perform the following steps. Notice that several steps might be applicable during one iteration.

- If $\sigma(F_s) = \sigma(B_t)$, we have found a codeword, thus, we increment *count*. If the reference *ref* is currently equal to zero, this is a possible beginning of a ‘cluster’ in the backward set, and we store the vertex number in *ref*, *i.e.*, $ref = t$.
- If $\sigma(F_s) < \sigma(B_t)$, we try to proceed with the next vertex in the backward set \mathcal{B}_{-j} , *i.e.*, we increase t by one. However, if

we cannot go on in the backward set because we have reached its end, we distinguish between two possibilities:

- (a) Previously we have found a match and the reference pointer $ref \neq 0$. Then we jump back with the currently active node B_t in the backward set \mathcal{B}_{-j} to the beginning of the cluster, *i.e.*, $t = ref$. Moreover, we proceed with the next vertex in the forward set \mathcal{F}_{+j} , *i.e.*, we increase s by one. This has to be done to avoid skipping possible matches at the end of one set if the other set already reached its end.

Additionally we stop, if it is not possible to proceed with the next vertex in the forward set \mathcal{F}_{+j} as we have also reached the end of the forward set.

- (b) The second option applies if $ref = 0$, *i.e.*, there was no previous match; then we stop.
 - If $\sigma(F_s) > \sigma(B_t)$ we try to proceed with the next vertex in the forward set \mathcal{F}_{+j} , *i.e.*, we increase s by one. If this is not possible we simply stop.

Otherwise, we check if the previous and the current vertex in the forward set \mathcal{F}_{+j} have the same state, *i.e.*, $\sigma(F_{s-1}) = \sigma(F_s)$.

If additionally, we have also stored the beginning of a ‘cluster’ in the reference pointer $ref \neq 0$, these two vertices in the forward set match with all the vertices in the actual ‘cluster’ of the backward set. In order to avoid skipping this additional matches, we jump back with the current active node B_t in the backward set \mathcal{B}_{-j} to the beginning of the cluster.

Using this algorithm the number of comparisons is reduced to at most $|\mathcal{F}_{+j}| + |\mathcal{B}_{-j}|$, as the vertices of both sets are processed roughly only once.

- Having calculated the number of codewords with weight w , it is not necessary to start from scratch in order to get the number of codewords with weight $(w + 1)$.

Increasing the weight w by one, either the *forward weight* f_w or the *backward weight* b_w changes. Thus either, the forward sets \mathcal{F}_{+j} or the backward sets \mathcal{B}_{-j} , $j = 0, 1, \dots, c - 1$ have to be updated, while the other set can be reused. We distinguish these two possibilities in the following:

- If the forward set has to be extended this can be accomplished in a straightforward way. According to (4.5) either all children ξ^C of one vertex ξ are stored in one of the c forward sets or none of them. Furthermore, as every forward set \mathcal{F}_{+j} contains only nodes of same weight $f_w + j$, the forward code tree simply starts to extend each entry, until (4.5) is fulfilled.
- If the backward set has to be extended, this requires more caution. First of all, according to (4.6) it is possible that only some of the children ξ^C of a given vertex ξ are stored in any of the c backward sets.

If we simply extend all current nodes in all c backward sets \mathcal{B}_{-j} , $j = 0, 1, \dots, c - 1$, until (4.6) is fulfilled, some nodes will be included several times. (Notice that for a given node ξ some of its children ξ^C are already stored in one set and thus they will be stored again during the extension.) To avoid this, it is necessary to keep track for every node ξ , which of its children ξ^C are already stored in one of the c current backward sets, in order to skip them during the extension.

Obviously, this can be only done efficiently for rate $R = b/c$ convolutional codes \mathcal{C} with small b , as this requires at least 2^b additional bits of storage for each node.

Example 4.1 The BEAST for a $R = 1/2$ convolutional code

Consider the rate $R = 1/2$ convolutional code whose encoding matrix is illustrated in Figure 2.3 and is given by

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

Assume we want to find the first two spectral components $n_{d_{\text{free}}}$ and $n_{d_{\text{free}}+1}$, using the BEAST as described above. Furthermore, we assume to have previously searched for codewords \mathbf{v} of weight $w = 0, 1, 2, 3, 4$ without success. Thus, we know that $d_{\text{free}} \geq 5$.

A possible distribution, searching for any codeword \mathbf{v} of weight $w = 5$ is given by $f_w = \lfloor w/2 \rfloor = 2$ and $b_w = \lceil w/2 \rceil = 3$.

The forward and backward tree for the weights f_w and b_w , respectively, is illustrated in Figure 4.4 and Figure 4.5. Following (4.5) all nodes in the forward tree with a weight $w = 2$ and $w = 3$ are stored, leading to the following two sets

$$\begin{aligned} \mathcal{F}_{+0} &= \{(1 \ 0)\} \\ \mathcal{F}_{+1} &= \emptyset \end{aligned}$$

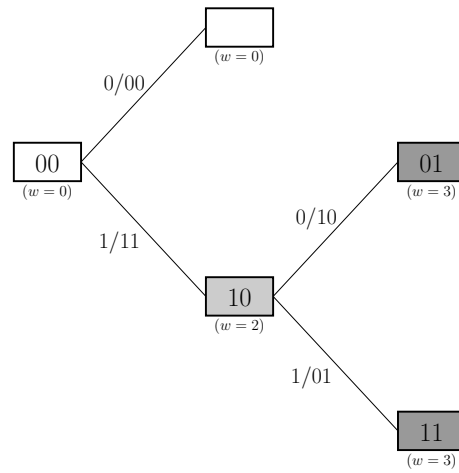


Figure 4.4: Forward tree explored by the BEAST with a forward weight $f_w = 2$ (light gray) and $f_w = 3$ (dark gray) for the rate $R = 1/2$ convolutional code from Figure 2.3 with encoding matrix $G(D) = \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}$.

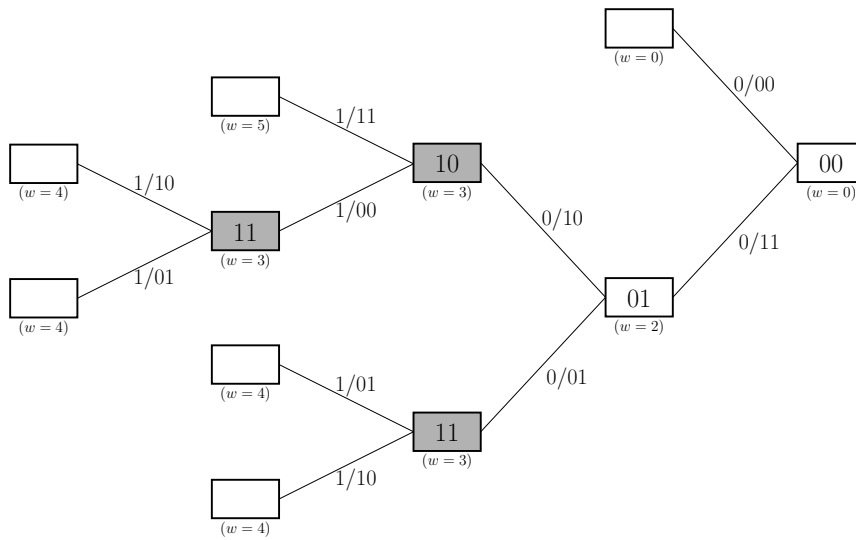


Figure 4.5: Backward tree explored by the BEAST with a backward weight $b_w = 3$ for the rate $R = 1/2$ convolutional code from Figure 2.3 with encoding matrix $G(D) = \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}$.

with the containing nodes marked in Figure 4.4 by light gray. For the backward tree, following (4.6), all nodes with weight $w = 3$ or $w = 2$ are stored in the corresponding backward sets

$$\begin{aligned}\mathcal{B}_{-0} &= \{(1\ 1), (1\ 1), (1\ 0)\} \\ \mathcal{B}_{-1} &= \emptyset\end{aligned}$$

marked by light gray in Figure 4.5. Notice that the vertices with empty states have not been calculated (visited), as it was sufficient to calculate the weight among the branches in order to determine to stop the extension in this direction.

Matching the sets \mathcal{F}_{+j} and \mathcal{B}_{-j} , leads to one single match, *i.e.*, there exists one codeword \mathbf{v} with weight $w = 5$. As there is no codeword of weight $w = 0, 1, \dots, 4$, the free distance d_{free} is given by $d_{\text{free}} = 5$ and the first spectral component $n_{d_{\text{free}}} = n_5 = 1$.

Going on and calculating the number of codewords \mathbf{v} of weight $w = 6$, the forward weight is increased by one to $f_w = 3$, while the backward weight remains the same, $b_w = 3$. Consequently, the backward sets \mathcal{B}_{-0} and \mathcal{B}_{-1} can be reused and only the forward sets have to be recalculated. Illustrated in Figure 4.4 in dark gray, the forward sets contain the following nodes,

$$\begin{aligned}\mathcal{F}_{+0} &= \{(0\ 1), (1\ 1)\} \\ \mathcal{F}_{+1} &= \emptyset\end{aligned}$$

with two matching codewords of weight $w = 6$. Thus, the second spectral component is given by $n_{d_{\text{free}}+1} = n_6 = 2$.

In this chapter, new rate $R = 1/2$ convolutional codes \mathcal{C} are presented, specified by their encoding matrices $G(D) = (g_1(D) \ g_2(D))$. Most of these convolutional codes have a larger free distance or at least a better spectrum compared to all previously known convolutional codes of same rate and memory. In order to retrieve such encoding matrices, their properties, presented in Chapter 3 and Chapter 4, are exploited.

As the first step, convolutional encoding matrices with an optimum free distance (OFD) obtained by an *exhaustive code search* are presented in Section 5.1. Afterwards, Section 5.2 introduces new optimum distance profile (ODP) convolutional encoding matrices based on the *optimum distance profile code search*.

5.1 Exhaustive Code Search

One way to find new rate $R = 1/2$ convolutional encoding matrices is the so-called *exhaustive code search*, where every possible combination of two encoding polynomials $g_1(D)$ and $g_2(D)$ is examined. For example, in the case of memory $m = 24$, there exist 2^{24} possible encoding polynomials for each, thus in total $2^{(2 \cdot 24)} = 2^{48}$ different encoding matrices $G(D)$ have to be inspected.

Although the BEAST, introduced in Section 4.2, is very fast and efficient, it is very time-consuming to check all encoding matrices by this algorithm. By implementing several efficient rejection rules, the number of encoding matrices $G(D)$ left for the inspection by the BEAST can be essentially reduced to $\varepsilon 2^{2m}$ with $\varepsilon \ll 1$.

For example, instead of 2^{48} possible encoding matrices $G(D)$ for a rate $R = 1/2$ convolutional code \mathcal{C} with memory $m = 24$, the number of encoding matrices which have to be checked by the BEAST, after applying the following rejection rules, can be reduced to 6830287, *i.e.*, $2 \cdot 10^{-9} \cdot 2^{48}$.

5.1.1 Rejection Rules

Rejection rules describe easy checkable conditions, which have to be satisfied as a precondition to obtain convolutional codes with good free distances d_{free} . We note that, obviously, the encoding matrices $G(D) = (g_1(D) \ g_2(D))$ and $G'(D) = (g_2(D) \ g_1(D))$ encode equivalent codes. Thus only one of them needs to be checked, reducing the number of possible encoding polynomials by 50%. The remaining encoding matrices are now examined using the following rejection rules:

- (i) Assume that we search for a rate $R = 1/2$ convolutional code \mathcal{C} with memory m . That means, at least one of the encoding polynomials $g_1(D)$ or $g_2(D)$ has to be of degree m (*cf.* (2.28)). As only half of all encoding polynomials have degree m , the number of encoding matrices $G(D) = (g_1(D) \ g_2(D))$, which have to be checked is further reduced by 25%.
- (ii) During this rejection rule we skip all so-called *catastrophic generator matrices*. When using such matrices to encode an information sequence \mathbf{u} with an unlimited number of 1s, it is possible to obtain an encoded sequence \mathbf{v} which contains a small number of 1s.

Encoding, *e.g.*, the information sequence $\mathbf{u} = 1 + D + D^2 + D^3 + \dots$ with weight $w_{\text{H}}(\mathbf{u}) = \infty$ with the catastrophic encoding matrix

$$G(D) = (1 + D^3 \quad 1 + D + D^2 + D^3)$$

yields the encoded sequence $\mathbf{v} = (1 \ 1) + (1 \ 0)D + (1 \ 1)D^2$ of weight $w_{\text{H}}(\mathbf{v}) = 5 < \infty$. A finite number of errors in the decision $\hat{\mathbf{v}} = \mathbf{0}$; in this case 5 errors at proper positions are sufficient for the receiver to decide for an information sequence $\hat{\mathbf{u}} = \mathbf{0}$ with an infinite number of errors.

Most of these catastrophic matrices can be easily skipped, checking if both encoding polynomials $g_1(D)$ and $g_2(D)$ have an even number of coefficients, *i.e.*, the polynomials are both divisible by $1 + D$. However, this does not cover all catastrophic matrices and thus an additional and more complex condition has to be applied on all remaining encoding matrices.

Consider the Smith form decomposition (*cf.* Section 3.4) $G(D) = A(D) \Gamma(D)B(D)$, where the diagonal elements of the $b \times b$ matrix $\Gamma(D)$ are either $\gamma_i(D)$, $i = 1, \dots, r$ or zero. According to [JZ98, Cor. 2.11], a polynomial convolutional generator matrix $G(D)$ is noncatastrophic if and only if $\gamma_b = D^s$ with any integer $s \geq 0$.

As we are considering only delay-free rate $R = 1/2$ encoding matrices, the matrix $\Gamma(D)$ contains only one factor $\gamma_1(D)$, which must be equal

m	$g_1(D)$	$g_2(D)$	d_{free}	spectrum	remark
12	53734	72304	* 16	14, 38, 35, 108, 342, 724	◇
13	63676	45272	16	1, 17, 38, 69, 158, 414	
14	75063	56711	* 18	26, 0, 165, 0, 845, 0	◇
15	533514	653444	19	30, 67, 54, 167, 632, 1402	◇
16	626656	463642	* 20	43, 0, 265, 0, 1341, 0	◇
17	611675	550363	20	4, 24, 76, 150, 354, 826	◇
18	4551474	6354344	22	65, 0, 349, 0, 1903, 0, 10947	◇
19	7504432	4625676	22	5, 52, 116, 163, 456, 1135	◇
20	6717423	5056615	* 24	145, 0, 225, 0, 3473, 0, 13659	◇
21	63646524	57112134	24	17, 95, 136, 138, 679, 2149	◇
22	64353362	41471446	25	47, 88, 137, 313, 912, 2172	◇
23	75420671	45452137	26	45, 0, 364, 0, 1968, 0, 10921	⊕
24	766446634	540125704	27	50, 135, 118, 294, 1481, 3299	⊗
25	662537146	505722162	28	71, 196, 112, 339, 2053, 4548	

Table 5.1: Convolutional encoding matrices $G(D) = (g_1(D) \ g_2(D))$ obtained by exhaustive code search.

to $\gamma_1(D) = D^0 = 1$ for noncatastrophic encoding matrices. For an encoding matrix $G(D) = (g_1(D) \ g_2(D))$ the factor $\gamma_1(D)$ can be obtained by calculating the greatest common divisor (gcd) of the encoding polynomials $g_1(D)$ and $g_2(D)$, and every generator matrix $G(D)$, for which $\gamma_1(D) \neq 1$ is catastrophic and, hence, skipped.

- (iii) The row distance d_j^r , introduced in Section 3.2, will be exploited as another rejection rule. We define a so-called *target free distance* $t_{d_{\text{free}}}$ and use the row distance as an upper bound on the free distance according to (3.17). An encoding matrix $G(D)$ is rejected as soon as at least one of its row distance values d_j^r , $j = 1, 2, \dots, n$, is smaller than the target free distance $t_{d_{\text{free}}}$. In other words, only encoding matrices $G(D)$ whose first row distance values d_j^r are greater than or equal to the target free distance $t_{d_{\text{free}}}$ will be considered further on, *i.e.*,

$$d_j^r \geq t_{d_{\text{free}}} \quad j = 0, 1, \dots, n. \quad (5.1)$$

Typically, we used $n \approx 20$ in our search.

5.1.2 Results

For all encoding matrices $G(D) = (g_1(D) \ g_2(D))$, that survived the rejection rules (i)–(iii), the free distance d_{free} is calculated by the BEAST. The best encoding matrices $G(D)$ for each memory m , together with their spectral components, are listed in Table 5.1 in *octal notation*, *e.g.*, $56 \stackrel{\text{def}}{=} 101 \ 110 \stackrel{\text{def}}{=} 1 + D^2 + D^3 + D^4$.

With our exhaustive code search, the results published in [BHJK04, Table II] and [JZ98, Table 8.3] were verified for most parts, marked by \diamond . The free distances highlighted by ‘*’ also fulfill the Griesmer bound, as, *e.g.*, given by [JZ98, Thm. 3.22]. However, the *optimum free distance (OFD)* encoding matrices up to memory $m = 24$, presented in [BHJK04], turned out not be entirely correct, since it was possible to improve some of those codes. For memory $m = 23$ an encoding matrix with a better spectrum was found, reducing the first spectral component from 51 to 45, indicated by ‘ \oplus ’. For memory $m = 24$ the free distance d_{free} was increased from 26 to 27, marked by ‘ \otimes ’. Furthermore, we were able to extend this table with an optimum free distance encoding matrix for memory $m = 25$.

5.2 Optimum Distance Profile Code Search

In case of an exhaustive code search according to Section 5.1 all possible rate $R = 1/2$ encoding matrices $G(D) = (g_1(D) \ g_2(D))$ with memory m are checked. Although after applying the rejection rules, only a small fraction of encoding matrices has to be checked by the BEAST, this is prohibitively complex for higher memories. By increasing the memory m by one, the computational time grows by a factor of four.

However, using the so-called *optimum distance profile (ODP) code search*, which is presented in this section, the computation time is reduced by limiting the code search to encoding matrices $G(D)$ satisfying the ODP property, which nevertheless results in a good free distance d_{free} .

5.2.1 Systematic Optimum Distance Profile Encoding Matrices

In the first step, we calculate systematic, ODP, rate $R = 1/2$ convolutional encoding matrices G_{sys} with at most memory m . These matrices can be used to obtain nonsystematic, ODP, rate $R = 1/2$ convolutional encoding matrices $G(D)$ with memory m as described later on in Section 5.2.2.

The distance profile \mathbf{d}^P , introduced in Section 3.10, is calculated for each of 2^m different encoding matrices $G_{\text{sys}}(D) = (1 \ g(D))$ and only those with an optimum distance profile (*cf.* (3.12)) are saved.

Let $\mathcal{G}_{\text{sys},m}$ denote the set of systematic, ODP, rate $R = 1/2$ encoding matrices $G_{\text{sys}}(D) = (1 \ g(D))$ with at most memory m . Notice that all encoding matrices in $\mathcal{G}_{\text{sys},m}$ have the same *minimum distance* $d_{\text{min}} \stackrel{\text{def}}{=} d_m^c$ but could differ in the number of (truncated) codewords $\mathbf{v}_{[0,m]}$ of weight d_{min} .

In Table 5.2 the number of matrices in $\mathcal{G}_{\text{sys},m}$, together with the minimum distance d_{min} , are listed for memory $m = 25, \dots, 40$. Additionally, for every memory m , a systematic, ODP, rate $R = 1/2$ encoding matrix with the fewest

m	$ \mathcal{G}_{\text{sys},m} $	d_{\min}	$\#\mathbf{v}_{[0,m]}$ of weight d_{\min}	$g(D)$
25	48	11	5	671145432
26	96	11	1	671145431
27	36	12	27	6711454574
28	72	12	8	6711454306
29	144	12	2	6711454306
30	12	13	43	67114545754
31	24	13	15	67114545754
32	48	13	4	67114545755
33	96	13	1	671145457554
34	12	14	34	671145457556
35	24	14	14	67114545447
36	48	14	5	6711454544704
37	96	14	2	6711454306444
38	16	15	31	6711454575564
39	32	15	12	6711454306444
40	64	15	3	67114545755712

Table 5.2: The number of and the minimum distance d_{\min} of systematic, ODP, rate $R = \frac{1}{2}$ encoding matrices for memory $m = 25, \dots, 40$. Additionally, one (among many), systematic, ODP, rate $R = \frac{1}{2}$ encoding matrix $G(D) = (1 \ g(D))$, specified by $g(D)$ with the fewest number of (truncated) codewords $\mathbf{v}_{[0,m]}$ of weight d_{\min} is presented.

number of (truncated) codewords $\mathbf{v}_{[0,m]}$ of weight d_{\min} is specified by $g(D)$. It has to be pointed out, that the specified encoding matrix $G(D) = (1 \ g(D))$ is only one encoding matrix out of the set $\mathcal{G}_{\text{sys},m}$ of systematic, ODP, rate $R = 1/2$ encoding matrices with memory m .

5.2.2 Nonsystematic Optimum Distance Profile Encoding Matrices

By exploiting the property mentioned in Section 3.1.1, every systematic ODP encoding matrix $G_{\text{sys}}(D) \in \mathcal{G}_{\text{sys},m}$ can be multiplied with any polynomial $t(D)$ of degree at most m and the product truncated to degree m , while the ODP property still remains. In other words, a nonsystematic, rate $R = 1/2$ encoding matrix can be obtained by multiplication with any polynomial $t(D)$ of degree at most m and truncating to degree m afterwards, *i.e.*,

$$G_{t(D)}(D) = (g_1(D) \ g_2(D)) = t(D)G_{\text{sys}}(D) \Big|_m \quad (5.2)$$

$$= \left(t(D) \ t(D)g(D) \Big|_m \right) \quad (5.3)$$

which still satisfies the ODP property. As $t(D)$ can be freely chosen as any polynomial of degree at most m , there exist 2^m different nonsystematic encoding matrices $G_{t(D)}(D)$ based on a single systematic encoding matrix $G_{\text{sys}}(D)$.

With the number of systematic encoding matrices $G_{\text{sys}}(D)$ given in Table 5.2, the total number of encoding matrices to be checked during the *optimum distance profile search* is upper-bounded by

$$2^m \cdot |\mathcal{G}_{\text{sys},m}(D)|.$$

Let $g(D)$ be any polynomial and $1/g(D)|_m$ be its inverse truncated to degree m . According to [JZ98, p. 112], for every systematic encoding matrix $G_{\text{sys}}(D) = (1 \ g(D)) \in \mathcal{G}_{\text{sys},m}$ with an optimum distance profile, there exists a truncated adjoint encoding matrix $G'_{\text{sys}}(D) = (1/g(D)|_m \ 1)$ with the same optimum distance profile. Thus, both encoding matrices are included in the same set $\mathcal{G}_{\text{sys},m}$.

Assume we are examining all nonsystematic, ODP, encoding matrices of the form

$$G_{t(D)}(D) = t(D) (1 \ g(D))|_m = (t(D) \ t(D)g(D)|_m) \quad (5.4)$$

with all possible polynomials $t(D)$ of degree at most m and $G(D) \in \mathcal{G}_{\text{sys},m}$. Then, as a by-product, we are also checking all nonsystematic, ODP, encoding matrices

$$G'_{t(D)}(D) = t(D) \left(\frac{1}{g(D)}|_m \ 1 \right) = \left(t(D) \frac{1}{g(D)}|_m \ t(D) \right) \quad (5.5)$$

obtained using the truncated adjoint encoding matrix $G'_{\text{sys}}(D)$. As we are iterating over all polynomials $t(D)$ with maximum degree m in (5.4), it is possible to replace $t(D)$ by

$$t(D) = \frac{t'(D)}{g(D)}|_m \quad (5.6)$$

with iterating over the polynomial $t'(D)$ with degree at most m . Replacing $t(D)$ in (5.4) by (5.6) yields directly (5.5). Thus, checking all possible nonsystematic, ODP encoding matrices obtained from $G_{\text{sys}} = (1 \ g(D))$, includes checking all nonsystematic, ODP encoding matrices obtained from the truncated adjoint encoding matrix $G'_{\text{sys}} = (1 \ 1/g(D)|_m)$. Thereby the number of remaining encoding matrices to be checked is reduced to $2^{m-1} \cdot |\mathcal{G}_{\text{sys},m}|$.

The same rejection rules, as described in Section 5.1, are applied to the remaining ODP encoding matrices. Thereby the number of ODP encoding matrices to be checked is further reduced, resulting in a small fraction of $\mu \cdot 2^{m-1} \cdot |\mathcal{G}_{\text{sys},m}|$, with $\mu \ll 1$. Only for these remaining nonsystematic, optimum distance profile, encoding matrices, the free distance d_{free} is finally calculated by the BEAST.

Comparing this number to the $\varepsilon \cdot 2^{2m}$, $\varepsilon \ll 1$ encoding matrices for *exhaustive code search* described in Section 5.1, the gain regarding computation time is obvious, especially for large memories m .

m	$g_1(D)$	$g_2(D)$	d_{free}	spectrum	remark
25	746411326	544134532	27	14, 58, 120, 264, 569, 1406	\oplus
26	525626523	645055711	28	24, 56, 131, 273, 736, 1723	\diamond
27	7270510714	5002176664	28	1, 28, 66, 138, 366, 789	\oplus
28	7605117332	5743521516	30	54, 0, 356, 0, 2148, 0	\diamond
29	7306324763	5136046755	30	5, 47, 97, 211, 514, 1171	\diamond
30	60425367524	45542642234	32	143, 0, 240, 0, 3870, 0	\otimes
31	51703207732	66455246536	32	14, 65, 136, 336, 753, 1860	\oplus
32	41273467427	70160662325	33	28, 61, 167, 372, 898, 2168	\otimes
33	407346436304	711526703754	34	44, 0, 338, 0, 2081, 0, 11973	\oplus
34	410174456276	702647441572	34	5, 35, 84, 229, 532, 1320	\oplus
35	627327244767	463171036121	36	111, 0, 553, 0, 3309, 0	\otimes
36	7664063056054	5707165143064	36	12, 53, 146, 360, 783, 1917	\oplus
37	7267577012232	5011131253046	37	18, 73, 163, 381, 884, 2232	\otimes
38	6660216760717	4131271202755	38	30, 83, 225, 524, 1152, 2761	\oplus
39	42576550101264	66340614757214	38	2, 38, 97, 219, 575, 1324	\oplus
40	26204724041271	37146123573117	40	78, 0, 532, 0, 6040, 0	\otimes

Table 5.3: Convolutional encoding matrices obtained by the optimum distance profile code search.

5.2.3 Results

For all nonsystematic, ODP, rate $R = 1/2$, encoding matrices $G(D)$ with memory m surviving the rejection rules, the free distance d_{free} is calculated by the BEAST. The best encoding matrices $G(D)$ for each memory m , together with their first spectral components, are listed in Table 5.3 in *octal notation*, e.g., $54 \stackrel{\text{def}}{=} 101\ 100 \stackrel{\text{def}}{=} 1 + D^2 + D^3$.

In [BHJK04, Table II] a list of optimum distance profile encoding matrices was presented. However, we have verified that not all of them have an optimum distance profile. The nonsystematic, ODP, rate $R = 1/2$ convolutional encoding matrices for memory $m = 26, 28, 29$ could be verified and the same results have been obtained, marked by ‘ \diamond ’. However, for memory $m = 25, 27, 31, 33, 34, 36, 38$ or 39 , nonsystematic, ODP, rate $R = 1/2$ encoding matrices with improved spectral components were found, highlighted by ‘ \oplus ’. Their free distance d_{free} remains the same compared to [BHJK04, Table II], while the number of codewords v , i.e., the spectral components decreases. Moreover, for memory $m = 30, 32, 35, 37$ and 40 it was possible to obtain nonsystematic, ODP, rate $R = 1/2$ encoding matrices with an improved free distance d_{free} , marked in Table 5.3 by ‘ \otimes ’.

Woven Graph Codes

As a second part of this thesis, *woven graph codes with constituent convolutional codes* [BKJZ07, BKJZ08, BKJZ] will be studied. Therefore Section 6.1 will introduce *graphs* and *hypergraphs*, while their representation in the form of corresponding incidence matrices is presented in Section 6.2. By associating different codes with the constraint nodes, *graph-based codes* and *hypergraph-based codes* are obtained. Based on these codes, Section 6.3 will finally introduce *woven graph codes with constituent convolutional codes*.

6.1 Graphs and Hypergraphs

A *graph* consists of several *vertices* and *edges* as connections between two different vertices, see Figure 6.1(a). However, if an edge connects more than two vertices, it is called *hyperedge* and the corresponding graph is said to be a *hypergraph*, see Figure 6.2. The number s of vertices connected by each hyperedge is called its *cardinality*. If the cardinality s is the same for every hyperedge, we denote the corresponding hypergraph as *s-uniform*. Clearly, in case of $s = 2$, the hypergraph is a graph.

The *degree of a vertex* in a hypergraph is the number of hyperedges connected to it. If all vertices have the same degree c , the hypergraph is *c-regular*, and c is the *degree of the hypergraph*.

Furthermore, assume that the set of all vertices \mathcal{V} of an s -uniform hypergraph is partitioned into t disjoint subsets \mathcal{V}_j , $j = 1, 2, \dots, t$. If no hyperedge connects two vertices from the same subset, the hypergraph is said to be *t-partite*.

Hereinafter, we only consider s -uniform, c -regular, s -partite hypergraphs, like the ones illustrated in Figure 6.2 and Figure 6.1(a). The 3-uniform, 4-regular, 3-partite hypergraph in Figure 6.2 consists of three subsets of vertices, each of size $n = 4$, shown as circles, diamonds, and squares, respectively.

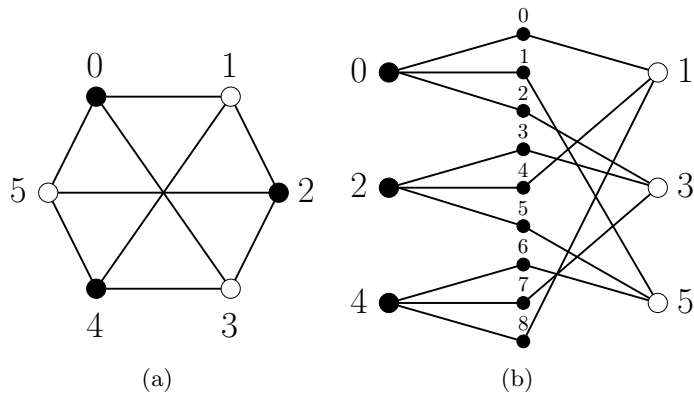


Figure 6.1: Utility bipartite, 2-uniform, 3-regular, 2-partite graph (a) and its representation as a Tanner graph (b).
 $(s = t = 2, c = 3 \text{ and } n = 3)$

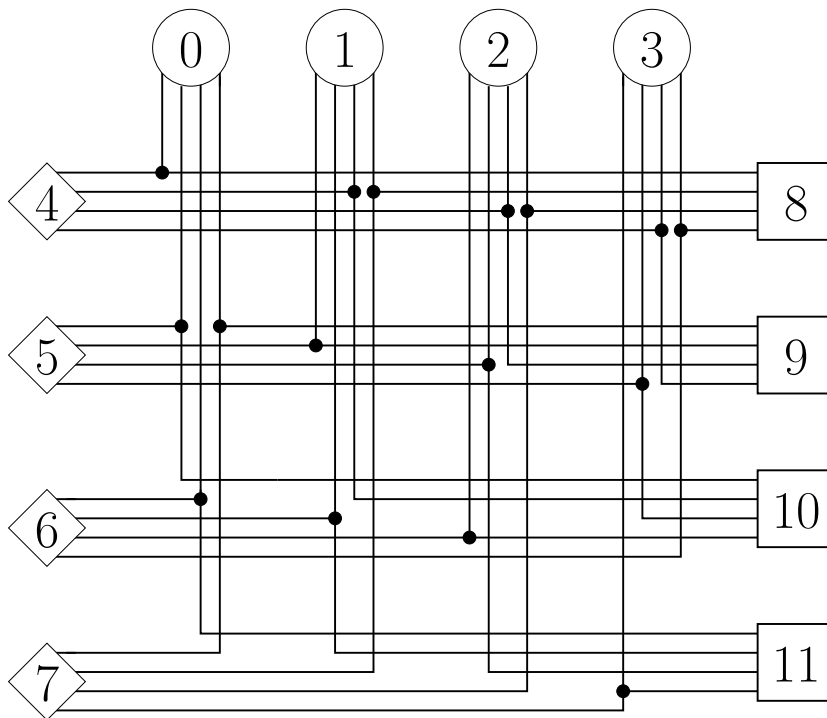


Figure 6.2: 3-uniform, 4-regular, 3-partite hypergraph.

Each vertex is connected to four different hyperedges, whereas each hyperedge connects three vertices from strictly disjoint sets \mathcal{V}_j , $j = 1, 2, 3$.

Another example is a 2-uniform, 3-regular, 2-partite graph, as illustrated in Figure 6.1(a). It consists of two disjoint sets of vertices \mathcal{V}_j , $j = 1, 2$ of size $n = 3$, indicated by circles in black and white, respectively. Each vertex of one set is connected to three different vertices from the other set, while every edge connects two vertices from disjoint sets, a ‘black’ and a ‘white’ vertex.

Cycle and Girth

According to [Die05, p. 7], a *cycle* of length L in the (hyper-)graph is an alternating sequence of L vertices and L (hyper-)edges, where all vertices and edges are distinct, except the initial and the final vertex, which coincide. The *girth* of a hypergraph is the length of its shortest cycle. For example, the graph illustrated in Figure 6.1(a) has a girth of 4; the sequence $0 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 0$ of 4 vertices and 4 edges is one of the shortest possible cycles.

6.2 Graph-Based Codes

The *incidence matrix* of the graph shown in Figure 6.1(a) is given by

$$H_{\text{gb}} = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 0 \\ 2 \\ 4 \\ 1 \\ 3 \\ 5 \end{matrix} & \left(\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{array} \right) \end{matrix} \quad (6.1)$$

where the columns correspond to edges and rows correspond to vertices, as indicated by the labels.

By replacing every edge of the utility bipartite graph in Figure 6.1(a) by a vertex with two outgoing edges, we obtain a so-called *Tanner graph* [Tan81], as illustrated in Figure 6.1(b). These newly introduced vertices will be called *symbol nodes*, whereas the original black and white vertices will be referred to as *constraint nodes*. Notice that this Tanner graph is bipartite, as it consists of two distinct sets of vertices, the symbol nodes and the constraint nodes, with no edge connecting two nodes from the same set.

Regarding the Tanner graph (Figure 6.1(b)), the columns of the incidence matrix (6.1) correspond to the symbol nodes, while rows correspond to the constraint nodes. Thus, a 1 at position (i, j) indicates a direct connection

between the i th constraint node and the j th symbol node, as illustrated in Figure 6.1(b).

By associating constituent codes with constraint nodes of the Tanner graph, and code symbols with its symbol nodes, a *graph-based code* is obtained. If the constraint nodes perform single parity-checks we obtain a rate $R = ((c - s)n) / (cn)$ graph-based code, whose parity-check matrix H_{gb} coincides with the incidence matrix (6.1). Every symbol node represents one of cn code symbols, while every row specifies one of sn parity-checks with the 1s indicating which code symbols participate in which parity-check. For example, the parity-check of row 1 in (6.1) involves the code symbols with the numbers 0, 4 and 8.

In case of the hypergraph illustrated in Figure 6.2, the corresponding incidence matrix is given by (6.2). Moreover, assuming single parity-checks to be performed in the constraint nodes, the parity-check matrix H_{hgb} for this *hypergraph-based code* coincides with its incidence matrix. Like for graph-based codes, the rows represent different parity-checks, while the columns correspond to the code symbols, with a 1 at position (i, j) indicating that the j th code symbol participates in the i th parity-check.

$$H_{\text{hgb}} = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \end{matrix} & \left(\begin{array}{cccc|cccc|cccc|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \right) . \end{matrix} \quad (6.2)$$

By reordering the rows of (6.2) in the following order $(0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11)$ the equivalent parity-check matrix H'_{hgb} is given by (6.3). Introducing the concept of the dual code \mathcal{C}^\perp [JZ98, Thm. 1.4], the equivalent parity-check matrix H'_{hgb} of this hypergraph-based code is equal to generator

matrix $G_{\text{hgb, tb}}^\perp$ of the corresponding dual code $\mathcal{C}_{\text{hgb, tb}}^\perp$.

$$G_{\text{hgb, tb}}^\perp = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{matrix} \\ \begin{matrix} 0 \\ 4 \\ 8 \\ 1 \\ 5 \\ 9 \\ 2 \\ 6 \\ 10 \\ 3 \\ 7 \\ 11 \end{matrix} & \left(\begin{array}{cccc|cccc|cccc|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \right) \end{matrix} \quad (6.3)$$

Obviously, this generator matrix $G_{\text{hgb, tb}}^\perp$ (6.3) of the rate $R = (sn)/(cn)$ dual code $\mathcal{C}_{\text{hgb, tb}}^\perp$ is *tail-bitten*. To illustrate this, (6.3) is expressed by 4 sub-matrices G_i^\perp , $i = 0, \dots, 3$, in (6.4), where its structure is obtained by tailbiting the semi-infinite generator matrix G_{hgb}^\perp (6.5) to length $L = 4$.

$$G_{\text{hgb, tb}}^\perp = \begin{pmatrix} G_0^\perp & G_1^\perp & G_2^\perp & G_3^\perp \\ G_3^\perp & G_0^\perp & G_1^\perp & G_2^\perp \\ G_2^\perp & G_3^\perp & G_0^\perp & G_1^\perp \\ G_1^\perp & G_3^\perp & G_1^\perp & G_0^\perp \end{pmatrix} \quad (6.4)$$

$$G_{\text{hgb}}^\perp = \begin{pmatrix} G_0^\perp & G_1^\perp & G_2^\perp & G_3^\perp & & & & \\ & G_0^\perp & G_1^\perp & G_2^\perp & G_3^\perp & & & \\ & & G_0^\perp & G_1^\perp & G_2^\perp & G_3^\perp & & \\ & & & G_0^\perp & G_1^\perp & G_2^\perp & G_3^\perp & \\ & & & & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \quad (6.5)$$

A short representation for this semi-infinite generator matrix G_{hgb}^\perp (6.5) of the dual code $\mathcal{C}_{\text{hgb}}^\perp$ in the time domain is given by its parent convolutional code $\mathcal{C}_{\text{hgb}}^{\perp p}$, specified by the generator matrix $G_{\text{hgb}}^{\perp p}(D)$.

$$G_{\text{hgb}}^{\perp p}(D) = G_0^\perp + G_1^\perp D + G_2^\perp D^2 + G_3^\perp D^3 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & D & D^2 & D^3 \\ 1 & D^3 & D & D^2 \end{pmatrix} \quad (6.6)$$

Similarly for the graph-based code (6.1), its parity-check matrix H_{gb} coincides with the generator matrix $G_{\text{gb, tb}}^\perp$ of the dual code $\mathcal{C}_{\text{gb, tb}}^\perp$. As the generator matrix $G_{\text{gb, tb}}^\perp$ is tailbitten, a parent convolutional code $\mathcal{C}_{\text{gb}}^{\perp p}$ can be

obtained, whose generator matrix $G_{\text{gb}}^{\perp\text{p}}(D)$ is given by

$$G_{\text{gb}}^{\perp\text{p}}(D) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & D & D^2 \end{pmatrix}. \quad (6.7)$$

By tailbiting the generator matrices (6.6) and (6.7) of the rate $R = s/c$ parent convolutional codes $\mathcal{C}_{\text{hgb}}^{\perp\text{p}}$ and $\mathcal{C}_{\text{gb}}^{\perp\text{p}}$, to length L , where L is any integer $L > m$, with memory m , different generator matrices with rate $R = (sL)/(cL)$ are obtained. In the following, these generator matrices are interpreted as incidence matrices of (hyper-)graphs, and are used to construct woven graph codes of different rates, which nevertheless are related to the same graph. Notice that the original incidence matrices (6.1) and (6.2) are obtained by tailbiting the generator matrices of the dual codes $\mathcal{C}_{\text{hg}}^{\perp}$ and $\mathcal{C}_{\text{hbg}}^{\perp}$ to length $L = m + 1$.

6.3 Woven Graph Codes with Constituent Convolutional Codes

By combining a (hyper-)graph-based code as introduced in Section 6.2 with a constituent convolutional code \mathcal{C}^c , a so-called *woven graph code with constituent convolutional code* can be constructed. Therefore, every (hyper-)edge leaving a constraint node, as illustrated in Figure 6.1(b) is labeled by a polynomial (or sub-matrix) of the parity-check matrix of the constituent rate $R = b^c/c^c$ convolutional code.

Consider a c -regular hypergraph, with c hyperedges leaving every constraint node and a rate $R = (c^c - 1)/c^c$ constituent convolutional code. The parity-check matrix $H^c(D)$ of such a constituent convolutional code consists of one row with c parity-check polynomials $h_i^c(D)$, $i = 1, \dots, c$, that is,

$$H^c(D) = (h_1^c(D) \quad h_2^c(D) \quad \dots \quad h_c^c(D)). \quad (6.8)$$

Consequently, each (hyper-)edge leaving a constraint node is labeled only by a single parity-check polynomial $h_i^c(D)$, $i = 1, \dots, c$. The (hyper-)edges leaving the constraint nodes on the left-hand side in Figure 6.1(b) are labeled successively by the constituent parity-check polynomials $h_i^c(D)$, $i = 1, \dots, c$, whereas the (hyper-)edges leaving the constraint nodes on the right-hand side are labeled by the same constituent parity-check polynomials but in a permuted order, which will be denoted by $t_i^c(D)$.

Notice that in general any constituent convolutional code \mathcal{C}^c of any rate $R = b^c/c^c$ can be used, as long as c^c is a multiple of c , so that its parity-check matrix can be separated in c sub-matrices of equal size $(c^c - b^c) \times (c^c/c)$, *i.e.*,

$$H^c(D) = (H_1^c(D) \quad H_2^c(D) \quad \dots \quad H_c^c(D)). \quad (6.9)$$

In this case, every (hyper-)edge leaving a constraint node is labeled by one of these c sub-matrices, instead of a single parity-check polynomial $h_i^c(D)$, and every symbol node represents a (c^c/c) -tuple of code symbols.

Consider the $s = 2$ uniform, $c = 3$ regular, $t = 2$ partite utility graph (Figure 6.1(a)). A suitable constituent convolutional code of rate $R = 1/3$ is given by the parity-check matrix

$$H^c(D) = \begin{pmatrix} h_1^c(D) & h_2^c(D) & h_3^c(D) \end{pmatrix}. \quad (6.10)$$

Then, the parity-check matrix $H_{\text{wg}}(D)$ of the woven graph code, based on the graph in Figure (6.1(b)), with a constituent convolutional code \mathcal{C}^c (6.10) is given by

$$H_{\text{wg}}(D) = \begin{array}{c} 0 \\ 2 \\ 4 \\ 1 \\ 3 \\ 5 \end{array} \left(\begin{array}{ccc|ccc|ccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline h_1^c(D) & h_2^c(D) & h_3^c(D) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_1^c(D) & h_2^c(D) & h_3^c(D) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_1^c(D) & h_2^c(D) & h_3^c(D) \\ \hline t_1^c(D) & 0 & 0 & 0 & t_2^c(D) & 0 & 0 & 0 & t_3^c(D) \\ 0 & 0 & t_3^c(D) & t_1^c(D) & 0 & 0 & 0 & t_2^c(D) & 0 \\ 0 & t_2^c(D) & 0 & 0 & 0 & t_3^c(D) & t_1^c(D) & 0 & 0 \end{array} \right) \quad (6.11)$$

where $t_1^c(D)$, $t_2^c(D)$, and $t_3^c(D)$ is one of six possible permutations of the polynomials $h_1^c(D)$, $h_2^c(D)$, and $h_3^c(D)$.

Such a code can be regarded as two dimensional, a tailbitten block code in one dimension and a convolutional code in the other dimension. According to [BKJZ], this is a special construction of a woven code [HJZ02] and is therefore called a *woven graph code*. The n constituent nodes on the left-hand side can be regarded as a warp with nc threads. Each of the n constituent nodes on the right-hand side is tacked on c of the threads in the warp, such that each thread of the warp is tacked on exactly once.

Consider now an s -uniform, c -regular, s -partite hypergraph in combination with a rate $R^c = b^c/c^c$ constituent convolutional code \mathcal{C}^c , whose parity-check matrix can be split into c sub-matrices of equal size $(c^c - b^c) \times (c^c/c)$ (cf. (6.9)). The total number of parity-checks is given by $sn(c^c - b^c)$, while the total number of code symbols is $c^c n$. Thus the rate R_{wg} of a woven graph code with constituent convolutional code is lower-bounded by

$$R_{\text{wg}} \geq \frac{(c^c - s(c^c - b^c))n}{c^c n} = 1 - s(1 - R^c). \quad (6.12)$$

Notice that the inequality is due to the fact that some of the parity-checks may be linearly dependent.

6.4 Encoding Matrices of Woven Graph Codes

To characterize a woven graph code with a constituent convolutional code \mathcal{C}^c , its free distance d_{free} has to be determined. A rough estimate can be obtained by using the free distance of the constituent code in combination with the girth (*cf.* Section 6.1) of the underlying graph according to [BKJZ].

Using the BEAST algorithm, introduced in Chapter 4, it is sometimes possible to determine the actual free distance d_{free} . Then the corresponding encoding matrix has to be calculated. A possible solution is given by (2.3) and (2.7), as the systematic encoding matrix

$$G_{\text{sys}}(D) = (I_K | R(D)) \quad (6.13)$$

and the corresponding systematic parity-check matrix

$$H(D) = (R(D)^T | I_{N-K}) \quad (6.14)$$

share the same, often rational, matrix $R(D)$.

As rational functions of two polynomials increase the computation complexity, it is much more convenient to avoid them. This can be easily accomplished by performing the following steps.

Starting from the parity-check matrix $H_{\text{wg}}(D)$ of a woven graph code and using only basic row operations, it is possible to obtain an equivalent $(c-b) \times c$ parity-check matrix $H'_{\text{wg}}(D)$, which can be written as

$$H'_{\text{wg}}(D) = (\text{diag}(d(D)) \quad P(D)). \quad (6.15)$$

The $(c-b) \times (c-b)$ diagonal matrix $\text{diag}(d(D))$ is given by

$$\text{diag}(d(D)) = d(D)I_{c-b} \quad (6.16)$$

where $d(D)$ is a polynomial, and $P(D)$ is a polynomial $(c-b) \times b$ matrix. Similarly to (2.3) and (2.7) a corresponding generator matrix can be written as

$$G(D) = (P(D)^T \quad \text{diag}(d(D))) \quad (6.17)$$

where now $\text{diag}(d(D))$ is of size $b \times b$.

Notice that this method for finding $G(D)$ is based on the same principle as when using the systematic encoding and parity-check matrices. However, by replacing the identity matrix I_{c-b} with the diagonal matrix $\text{diag}(d(D)) = d(D)I_{c-b}$, the (possibly) rational matrix $R(D)$ can be replaced by a polynomial matrix $P(D)$.

It has to be noted that, in general, the obtained generator matrix given by (6.17) does not fulfill the minimal-basic property. Thus, an equivalent

minimal-basic encoding matrix in minimal span form $G_{\text{mbms}}(D)$ is calculated using the Smith form decomposition, the basic, minimal-basic, and minimal span concepts (*cf.* Sections 3.4 - 3.6). Thereby we have obtained an encoding matrix, suitable for finding the actual free distance d_{free} using the BEAST.

Promising Examples of Woven Graph Codes

In this chapter, two interesting examples of woven graph codes with constituent convolutional codes will be presented. Starting with the parity-check matrix $H_{\text{wg}}(D)$ of the woven graph code, the corresponding minimal-basic encoding matrix $G_{\text{wg,mbms}}(D)$ in its minimal span form will be obtained. Using the BEAST and the first row distance values d_j^r , the free distance d_{free} will be specified as far as possible. While for the first example we will only determine an interval for the free distance, a definite value of 120 will be obtained for the second one.

7.1 Example 1

A set of different incidence matrices for hypergraphs is given by the rate $R = 3/6$ generator matrix

$$G^{\perp\text{p}}(Z) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & Z & Z^2 & Z^3 & Z^4 & Z^5 \\ 1 & Z^5 & Z^4 & Z^3 & Z^2 & Z^1 \end{pmatrix} \quad (7.1)$$

of the parent convolutional code $\mathcal{C}^{\perp\text{p}}$, tailbitten to different lengths L . The parent convolutional code $\mathcal{C}^{\perp\text{p}}$ itself is obtained as the parent code of the dual code \mathcal{C}^{\perp} , whose generator matrix G^{\perp} tailbitten to length m is equal to the incidence matrix of the originally underlying hypergraph.

A woven graph code with a constituent convolutional code may be regarded as a two-dimensional code. The first dimension is obtained using the incidence matrix specified by the generator matrix of the parent convolutional code $\mathcal{C}^{\perp\text{p}}$ tailbitten to length L , while the second dimension is given by the constituent convolutional code \mathcal{C}^{c} . In order to distinguish between those two

dimensions the first one will be denoted by the variable Z , while D will be used for the latter one.

By tailbiting (7.1) to length $L = 6$, an 18×36 generator matrix is obtained, which may be interpreted as the incidence matrix of a hypergraph. We use a rate $R = 5/6$ constituent convolutional code \mathcal{C}^c specified by the parity-check matrix

$$H^c(D) = \begin{pmatrix} 1 + D + D^2 + D^3 + D^5 \\ 1 + D + D^2 + D^5 \\ 1 + D + D^2 + D^4 + D^5 \\ 1 + D + D^2 + D^3 + D^5 \\ 1 + D^3 + D^5 \\ 1 + D + D^5 \end{pmatrix}^T = \begin{pmatrix} h_1^c(D) \\ h_2^c(D) \\ h_3^c(D) \\ h_4^c(D) \\ h_5^c(D) \\ h_6^c(D) \end{pmatrix}^T. \quad (7.2)$$

The parity-checks are assigned to the constraint nodes of this hypergraph, and the parity-check matrix $H_{\text{wg}}(D)$ of the rate $R = 18/36$ woven graph code with constituent convolutional code \mathcal{C}^c is obtained as given by (7.8). The chosen permutations for the second and third set of vertices are specified by the vectors $\mathbf{t}^c(D)$ and $\mathbf{l}^c(D)$,

$$\begin{aligned} \mathbf{t}^c(D) &= (t_1^c(D) \ t_2^c(D) \ t_3^c(D) \ t_4^c(D) \ t_5^c(D) \ t_6^c(D)) \\ \mathbf{l}^c(D) &= (l_1^c(D) \ l_2^c(D) \ l_3^c(D) \ l_4^c(D) \ l_5^c(D) \ l_6^c(D)) \end{aligned}$$

with the individual polynomials $t_i^c(D)$ and $l_i^c(D)$, $i = 1, \dots, 6$, being mapped to $h_i^c(D)$ by

$$t_i^c(D) = h_{(i \bmod 6)+1}^c(D) \quad (7.3)$$

$$l_i^c(D) = h_{((i+1) \bmod 6)+1}^c(D) \quad (7.4)$$

where \bmod denotes the modulo operator. Notice that there may also exist other suitable permutations, yielding even larger free distances.

It is theoretically possible to obtain an equivalent parity-check matrix $H'_{\text{wg}}(D)$ in the form of (6.15) by only using basic row operations as suggested in Section 6.4. In practical implementations, however, this leads to intermediate polynomials with a maximum degree $d > 8000$, handled by implementing this process dynamically (*cf.* Appendix A.1 and Appendix A.3). This means, the memory needed for storing the polynomial coefficients is allocated dynamical during runtime and we finally obtain the corresponding equivalent generator matrix $G_{\text{wg}}(D)$. Notice that, in general, the generator matrix $G_{\text{wg}}(D)$ obtained according to Section 6.4 is not the minimal-basic. Therefore, the equivalent minimal-basic encoding matrix in its minimal span form $G_{\text{wg,mbms}}(D)$ is calculated using the Smith form decomposition, the basic, minimal-basic, and minimal span form concepts (*cf.* Sections 3.4 - 3.6).

For example, the equivalent minimal-basic encoding matrix $G_{\text{wg,mbms}}(D)$ in its minimal span form with memory $m = 6$ and overall constraint length $\nu = 84$ is given by (7.9), with the polynomials written in *octal notation*, e.g., $54 \stackrel{\text{def}}{=} 101\ 100 \stackrel{\text{def}}{=} 1 + D^2 + D^3$.

Using $G_{\text{wg,mbms}}(D)$ we have tried to find its free distance d_{free} . Using the BEAST, introduced in Section 4.2, to successively check for codewords \mathbf{v} of weight $w = 1, 2, 3, \dots$, we have obtained a lower bound for d_{free} . Moreover, calculating the first row distance values d_j^r , $j = 0, 1, 2, \dots$, following Section 3.2, an upper bound on d_{free} could be obtained.

At the time this thesis was printed, we have not found the free distance d_{free} . However we are able to specify the following interval for the free distance

$$28 \leq d_{\text{free}} \leq 60. \quad (7.5)$$

7.2 Example 2

As a second example we will present a rate $R = 5/20$ woven graph code with a constituent convolutional code constructed according to Section 6.3. Let us consider the set of incidence matrices for hypergraphs, given by the rate $R = 3/4$ generator matrix $G^{\perp\text{p}}$ (7.6) of the the parent convolutional code $\mathcal{C}^{\perp\text{p}}$. This parent convolutional code $\mathcal{C}^{\perp\text{p}}$ is obtained as the parent code of the dual code \mathcal{C}^{\perp} , whose tailbitten generator matrix G^{\perp} is equal to the incidence matrix of the underlying hypergraph given in Figure 6.2 (*cf.* Section 6.2).

$$G^{\perp\text{p}}(Z) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & Z & Z^2 & Z^3 \\ 1 & Z^3 & Z & Z^2 \end{pmatrix}. \quad (7.6)$$

By tailbiting (7.6) to length $L = 5$, we obtained a rate $R = 15/20$ generator matrix, which can be interpreted as the incidence matrix of a hypergraph with 20 vertices and 15 edges or 20 constraint nodes and 15 symbol nodes, respectively. In order to obtain the parity-check matrix of the woven graph code, the constraint nodes are associated with the parity-checks of the constituent convolutional code \mathcal{C}^c with rate $R^c = 3/4$, determined by the parity-check matrix

$$H^c(D) = \begin{pmatrix} 1 + D + D^2 + D^4 \\ 1 + D + D^2 + D^3 + D^4 \\ 1 + D + D^3 + D^5 \\ 1 + D^2 + D^5 \end{pmatrix}^T = \begin{pmatrix} h_1^c(D) \\ h_2^c(D) \\ h_3^c(D) \\ h_4^c(D) \end{pmatrix}^T. \quad (7.7)$$

Encoding matrix of a rate $R = 18/36$ woven graph code

$$C_{\text{wgm.bms}}(D) = \begin{pmatrix} 54 & 54 & 24 & 3 & 24 & 2 & 54 & 54 & 24 & 3 & 24 & 2 & 54 & 54 & 24 & 3 & 24 & 2 & 54 & 54 & 24 & 3 & 24 & 2 \\ 3 & 5 & 1 & 0 & 34 & 44 & 3 & 5 & 1 & 0 & 34 & 44 & 3 & 5 & 1 & 0 & 34 & 44 & 3 & 5 & 1 & 0 & 34 & 44 \\ 06 & 2 & 42 & 12 & 2 & 52 & 74 & 46 & 74 & 4 & 34 & 22 & 32 & 3 & 52 & 44 & 76 & 46 & 14 & 4 & 76 & 12 & 46 & 42 \\ 3 & 06 & 26 & 42 & 16 & 5 & 2 & 06 & 34 & 36 & 1 & 34 & 7 & 62 & 06 & 04 & 06 & 12 & 54 & 72 & 04 & 26 & 56 & 62 \\ 32 & 3 & 24 & 02 & 54 & 4 & 32 & 02 & 04 & 62 & 04 & 42 & 36 & 42 & 04 & 34 & 66 & 12 & 66 & 14 & 16 & 66 & 2 & 36 \\ 04 & 24 & 04 & 02 & 1 & 36 & 4 & 0 & 24 & 12 & 62 & 24 & 64 & 22 & 06 & 5 & 62 & 56 & 5 & 0 & 36 & 66 & 64 & 4 \\ 26 & 32 & 0 & 14 & 3 & 2 & 16 & 62 & 1 & 14 & 04 & 64 & 26 & 32 & 0 & 14 & 3 & 2 & 16 & 62 & 1 & 14 & 04 & 64 \\ 33 & 13 & 37 & 13 & 03 & 05 & 31 & 0 & 53 & 32 & 14 & 56 & 47 & 03 & 57 & 66 & 21 & 52 & 45 & 66 & 67 & 26 & 16 & 66 \\ 07 & 21 & 24 & 32 & 13 & 23 & 17 & 17 & 24 & 71 & 36 & 63 & 03 & 34 & 13 & 33 & 07 & 12 & 0 & 67 & 72 & 77 & 52 & 1 \\ 24 & 36 & 22 & 34 & 22 & 32 & 0 & 26 & 2 & 16 & 72 & 42 & 3 & 44 & 1 & 76 & 16 & 04 & 04 & 54 & 0 & 2 & 4 & 1 \\ 13 & 11 & 17 & 32 & 05 & 24 & 0 & 15 & 35 & 15 & 32 & 11 & 72 & 23 & 26 & 61 & 7 & 46 & 2 & 7 & 21 & 32 & 03 & 56 \\ 35 & 21 & 35 & 13 & 01 & 35 & 36 & 31 & 3 & 27 & 33 & 33 & 3 & 47 & 66 & 0 & 05 & 14 & 02 & 57 & 74 & 1 & 45 & 4 \\ 34 & 06 & 27 & 22 & 25 & 0 & 24 & 24 & 26 & 24 & 03 & 01 & 04 & 15 & 75 & 1 & 32 & 44 & 43 & 62 & 73 & 37 & 0 & 73 \\ 14 & 23 & 13 & 26 & 2 & 34 & 17 & 05 & 31 & 33 & 32 & 3 & 22 & 31 & 36 & 67 & 23 & 65 & 4 & 2 & 44 & 01 & 71 & 74 \\ 154 & 164 & 024 & 304 & 374 & 114 & 24 & 36 & 23 & 0 & 21 & 02 & 124 & 034 & 114 & 014 & 414 & 714 & 67 & 0 & 03 & 14 & 56 & 04 \\ 244 & 244 & 114 & 374 & 114 & 364 & 024 & 264 & 364 & 304 & 134 & 324 & 2 & 16 & 35 & 33 & 11 & 23 & 65 & 43 & 57 & 66 & 44 & 61 \\ 32 & 044 & 33 & 054 & 154 & 134 & 154 & 334 & 054 & 37 & 314 & 36 & 374 & 05 & 274 & 3 & 07 & 27 & 2 & 15 & 46 & 03 & 44 & 14 \\ 004 & 03 & 32 & 174 & 014 & 214 & 24 & 11 & 004 & 074 & 17 & 26 & 03 & 164 & 114 & 264 & 124 & 33 & 124 & 254 & 37 & 02 & 2 & 36 \end{pmatrix} \quad (7.9)$$

f_w / b_w	Number of vertices
forward 50	1214218602
forward 51	1779808053
forward 52	2609654183
forward 53	3829478763
forward 54	5621507853
forward 55	8249059638
forward 56	12101010797
forward 57	17747837497
backward 50	1680555555
backward 55	797407021
backward 60	7277041432
backward 62	15205251727

Table 7.1: The number of stored vertices in some unions of forward sets \mathcal{F}_{+j} with weight $f_w + j$, $j = 0, 1, \dots, (c-1)$ and backward sets \mathcal{B}_{-j} with weight $b_w - j$, $j = 0, 1, \dots, (c-1)$, used during the calculation of the free distance d_{free} .

Moreover, the permutation matrices $\mathbf{t}^c(D)$ and $\mathbf{l}^c(D)$ for the second and third set of vertices \mathcal{V}_t , $t = 2, 3$, respectively, are given by

$$\begin{aligned} \mathbf{t}^c(D) &= \begin{pmatrix} t_1^c(D) & t_2^c(D) & t_3^c(D) & t_4^c(D) \\ l_1^c(D) & l_2^c(D) & l_3^c(D) & l_4^c(D) \end{pmatrix} \\ \mathbf{l}^c(D) &= \begin{pmatrix} l_1^c(D) & l_2^c(D) & l_3^c(D) & l_4^c(D) \end{pmatrix}. \end{aligned}$$

One of several suitable permutations is determined by

$$\begin{aligned} h_1^c(D) &= t_1^c(D) = l_3^c(D) = 1 + D + D^2 + D^4 \\ h_2^c(D) &= t_4^c(D) = l_4^c(D) = 1 + D + D^2 + D^3 + D^4 \\ h_3^c(D) &= t_2^c(D) = l_1^c(D) = 1 + D + D^3 + D^5 \\ h_4^c(D) &= t_3^c(D) = l_2^c(D) = 1 + D^2 + D^5. \end{aligned}$$

Associating the parity-checks of the constituent code to the constraint nodes of the hypergraph, for the second and third set of vertices according to their permutation matrices, the parity-check matrix $H_{\text{wg}}(D)$ of the rate $R = 5/20$ woven graph code is obtained, which is illustrated in (7.8).

Following Section 6.4, a corresponding generator matrix $G_{\text{wg}}(D)$ could be obtained. In order to reduce the complexity of finding its free distance d_{free} by the BEAST, the equivalent minimal-basic encoding matrix in its minimal span form $G_{\text{wg,mbms}}(D)$ given by (7.11) is calculated (*cf.* Sections 3.4 - 3.6).

The rate $R = 5/20$ minimal-basic encoding matrix for this woven graph code has memory $m = 14$ and overall constraint length $\nu = 67$. Notice that

the polynomials are given in *octal notation*, e.g., $4463 \stackrel{\text{def}}{=} 100\ 100\ 110\ 011 \stackrel{\text{def}}{=} 1 + D^3 + D^6 + D^7 + D^{10} + D^{11}$.

By using the BEAST (*cf.* Section 4.2), we have tried to determine the free distance d_{free} of the rate $R = 5/20$ woven graph code \mathcal{C} by successively checking for codewords \mathbf{v} of weight $w = 1, 2, 3, \dots$

Using a calculation cluster, $c = 20$ forward sets \mathcal{F}_{+j} and 20 backward sets \mathcal{B}_{-j} , $j = 0, 1, \dots, 19$, have been calculated and sorted separately for different weights w . The forward and backward sets, together with the total number of vertices for all 20 sets are listed in Table 7.1. Following (4.4) any distribution of the forward weight f_w and backward weight b_w is possible, thus, by matching each forward set with all backward sets, several weights w could be checked at once, e.g., $w = 100, 101, \dots, 119$, are checked using only the sets listed in Table 7.1.

In addition to checking for codewords \mathbf{v} of weight w by the BEAST, the first row distance values d_j^r , $j = 0, 1, 2, \dots$, have been calculated, to obtain an upper bound on the free distance d_{free} , as $d_{\text{free}} \leq d_j^r$ (*cf.* Section (3.2)). Finally, we have verified the free distance d_{free} for this rate $R = 5/20$ woven graph code to be 120.

Encoding matrix of a rate $R = 5/20$ woven graph code

$$G_{\text{wg,mbms}}(D) = \begin{pmatrix} 4463 & 7413 & 6523 & 6153 & 4463 & 7413 & 6523 & 6153 & 4463 & 7413 & 6523 & 6153 & 4463 & 7413 & 6523 & 6153 \\ \hline 1473 & 40453 & 16256 & 62224 & 44364 & 50324 & 36077 & 30173 & 53717 & 4266 & 30434 & 32352 & 37464 & 14262 & 6517 & 71254 & 47726 & 14624 & 31724 & 5234 \\ \hline 35532 & 16313 & 70256 & 612 & 3617 & 52613 & 77505 & 03033 & 71157 & 5042 & 51367 & 53145 & 02657 & 14566 & 0226 & 12262 & 16666 & 42522 & 0317 & 7227 \\ \hline 31304 & 23103 & 32474 & 34526 & 42404 & 67045 & 61573 & 75601 & 56435 & 31001 & 60663 & 15613 & 75761 & 6563 & 13545 & 06663 & 76323 & 2133 & 14216 & 6445 \\ \hline 17372 & 04153 & 11505 & 00017 & 03343 & 52117 & 10615 & 60005 & 20017 & 06726 & 63533 & 73075 & 23455 & 42226 & 6426 & 11246 & 64472 & 40015 & 42402 & 4133 \end{pmatrix} \quad (7.11)$$

Conclusions

Error correcting capabilities of a convolutional code are predominantly determined by its free distance and its spectrum. The first part of this thesis was devoted to the search for codes with larger free distances and better spectra than previously known. In order to find such convolutional codes, several distance properties were exploited. The row distance was used to reduce the number of possible encoding matrices during the exhaustive code search up to memory 25 and during the optimum distance profile code search for memory 25 to memory 40. For all encoding matrices surviving this and other rejection rules, the free distance was calculated by the BEAST — Bidirectional Efficient Algorithm for Searching code Trees.

We highlight the following results: new rate $1/2$ convolutional codes with optimum free distance for memory 24 (free distance 27) and memory 25 (free distance 28) were found. While searching for convolutional codes with an optimum distance profile, codes with larger free distances were found for memories 30, 32, 35, 37, and 40, while for memories 25, 27, 31, 33, 34, 36, and 38 the found codes provide a better spectrum than that of all codes previously known.

The second part of the thesis covers the recently introduced convolutional codes called woven graph codes. These codes are constructed by combining the incidence matrices of a (hyper-)graph with the parity-checks of a constituent convolutional code. Exploiting their structural properties the corresponding generator matrix was obtained. By performing additional steps an equivalent minimal-basic encoding matrix in its minimal-span form was found.

Promising examples of woven graph codes were presented and the BEAST was used to successively check for the nonexistence of codewords of certain weights as a lower bound. With the row distances serving as an upper bound, we successfully reduced the interval for the free distances of these codes. Thereby, the free distance for a rate $5/20$ woven graph code was finally calculated to be 120. This emphasizes the good potential of woven graph codes, since they have large free distances and they are iteratively decodable.

Outlook

The algorithms for the exhaustive code search and the optimum distance profile code search are well designed and can be used to obtain further encoding matrices with higher memories. Using highly efficient computational clusters it should be possible to extend the search to very large memories.

For the newly introduced woven graph codes, the free distance depends to a large extent on the chosen permutations of the parity-check of the constituent code. It would be of interest to determine which permutation leads to the largest free distance. Moreover, finding efficient iterative decoding schemes for woven graph codes is necessary in order to be able to use these codes in practical applications.

References

- [BHJK01] I.E. Bocharova, M. Handlery, R. Johannesson, and B.D. Kudryashov. A beast for prowling in trees. In *Proc. 39th Annual Allerton Conf. Commun., Control, and Computing*, Monticello, Illinois, USA, October 2001.
- [BHJK04] I. Bocharova, M. Handlery, R. Johannesson, and B.D. Kudryashov. A beast for prowling in trees. *IEEE Trans. Inform. Theory*, 50(6):1295–1302, June 2004.
- [BKJZ] Irina E. Bocharova, Boris D. Kudryashov, Rolf Johannesson, and Victor V. Zyablov. Woven graph codes: Asymptotic performances and examples.
- [BKJZ07] Irina E. Bocharova, Boris D. Kudryashov, Rolf Johannesson, and Victor V. Zyablov. Asymptotically good woven codes with fixed constituent convolutional codes. *IEEE International Symposium on Information Theory*, June 2007.
- [BKJZ08] Irina E. Bocharova, Boris D. Kudryashov, Rolf Johannesson, and Victor V. Zyablov. Woven graph codes over hyper graphs. *7th International ITG Conference on Source and Channel Coding*, January 2008.
- [boo] Boost c++ libraries.
- [Bos98] Martin Bossert. *Kanalcodierung*. Teubner, Stuttgart, 1998.
- [Cos69] D.J. Costello, Jr. A construction technique for random-error-correcting convolutional codes. *IEEE Trans. Inform. Theory*, 19:631–636, 1969.

- [Die05] Reinhard Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 3rd. edition, July 2005.
- [Eli55] Peter Elias. Coding for noisy channels. *Proc. IRE Conf. Record*, 4:37–46, 1955.
- [For70] G.D. Forney, Jr. Convolutional codes I: Algebraic structure. *IEEE Trans. Inform. Theory*, 16:720–738, 1970.
- [For88] G.D. Forney, Jr. Coset codes - part II: Binary lattices and related codes. *IEEE Trans. Inform. Theory*, 34:1049–1053, September 1988.
- [HJZ02] S. Höst, R. Johannesson, and V. V. Zyablov. Woven convolutional codes I: Encoder properties. *IEEE Trans. Inform. Theory*, 48(1):149–161, January 2002.
- [JZ98] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE Press, 1998.
- [Lon07] Maja Lončar. *Taming of the BEAST*. PhD thesis, Lund University, 2007.
- [Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July and October 1948.
- [Tan81] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27(5):553–547, September 1981.
- [Vit67] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, 13:260–269, 1967.

Software Toolbox

During this thesis a *software toolbox*, providing all necessary algorithms, was developed. In order to achieve efficient implementation, we decided to use the C++ programming language together with the *Boost C++ Libraries* [boo]. In the following the different parts of this software toolbox will be presented.

A.1 Binary Polynomials

A C++ class named *binary_poly* was developed to implement the metric polynomials with binary coefficients. These polynomials can be represented by their decimal value, octal value or as a sequence of characters consisting of 0s and 1s. No upper limit is imposed on the degree of these binary polynomials, as necessary memory is allocated dynamically during runtime. Thereby it was possible to perform calculations with binary polynomials of degree greater than 8000.

This class provides some basic operations like *addition*, *subtraction*, *multiplication* and *division*, but also logical operators like *and*, *or* and *xor* as so-called overloaded functions. This means they can be used in their natural context without having to call special functions. Moreover, the individual polynomial coefficient can be accessed directly, *e.g.*, by functions for obtaining the lowest as well as the highest degree and counting the number of coefficients.

Finally, so-called *helper functions* implement well-known algorithms like the *extended Euclidean algorithm* or variations like *greatest common divisor* or *division with remainder*.

A.2 Vertex Container

Another C++ class, implemented in several variants, provides an intelligent container for storing vertices for the BEAST (*cf.* Section 4.2). This class accepts an unlimited amount of vertices consisting of the state, the depth and the weight. This information is stored properly and returned upon request by this class. In order to provide an efficient algorithm, the vertices are stored initially in the memory (if not specified otherwise), but as soon as a certain threshold is exceeded, it will be switched to files on a specified hard disc.

For the algorithm itself, *e.g.*, the BEAST, there is no obvious difference how the vertices are stored, as all the implementation logic and operations are transparently performed by the vertex container class.

A.3 Minimal-Basic, Minimal-Span

In order to calculate the corresponding minimal-basic, minimal-span encoding matrix $G_{\text{wg,mbms}}$ for a given parity-check matrix H_{wg} the application *minimal-basic, minimal-span (form) (MBMSF)* was developed. Following Section 6.4, for a given parity-check matrix H_{wg} , a corresponding generator matrix G_{wg} is obtained. However, in general, such a generator matrix does not have the minimal-basic and/or minimal-span form. By decomposing this matrix to the Smith form (*cf.* Section 3.4), we get an equivalent basic encoding matrix (*cf.* Section 3.5.1). Following Section 3.5.2 and Section 3.6 a minimal-basic, minimal-span encoding matrix $G_{\text{wg,mbms}}$ is finally derived. Notice that during this procedure, some of the intermediate polynomials may have an extremely high degree (≥ 8000). However, by using the *binary_poly* class (*cf.* Section A.1) this can be handled easily.

A.4 Optimum Distance Profile

An application called *Optimum Distance Profile (ODP)* was developed to find systematic rate $R = 1/2$ convolutional encoders with an optimum distance property, introduced in Section 3.1.1. Therefore a search over all possible combinations is performed while only the best ones are stored.

A list of all rate $R = 1/2$ systematic encoding matrices with memory $m = 1, 2, \dots$ was calculated, being reused later on during the *Optimum Distance Profile Code Search*, which will be described in Section A.5.

Moreover, for any given rate $R = 1/2$ convolutional code \mathcal{C} the corresponding distance profile $\mathbf{d}^{\mathcal{P}}$,

$$\mathbf{d}^{\mathcal{P}} = (d_0^{\mathcal{C}}, d_1^{\mathcal{C}}, d_2^{\mathcal{C}}, \dots, d_m^{\mathcal{C}})$$

can be calculated according following (3.10). However, notice that this implementation is currently covers only rate $R = 1/2$ convolutional codes \mathcal{C} with any memory m .

A.5 Search

Exhaustive Code Search (cf. Section 5.1) and *Optimum Distance Profile Code Search* (cf. Section 5.2) is implemented by a further application (*SEARCH*).

For exhaustive code search, two nested iteration loops check all possible polynomials with a degree not smaller than the memory m . Every combination is checked by the rejection rules, discussed in Section 5.1.1, while the remaining polynomials are stored for further processing by the BEAST (cf. Section 4.2).

In case of an optimum distance profile (ODP) code search, the list of the systematic rate $R = 1/2$ optimum distance encoding matrices, obtained by the *ODP* application (cf. Section A.4) is reused. Based on Section 3.1.1, several nonsystematic encoding matrices are obtained and checked against the same rejection rules as for the exhaustive code search. All nonsystematic encoding matrices passing these checks are stored for determining their free distance d_{free} by the BEAST (cf. Section 4.2).

This implementation covers all $R = 1/2$ convolutional codes \mathcal{C} with memory m , with the maximum row distance d_j^r being upper bounded by $m + j_{\text{max}} < 64$.

According to (3.14), the row distance value d_j^r is defined as the minimum weight of all codewords \mathbf{v} obtained by using an information sequence \mathbf{u} of a certain length. In case of a rate $R = 1/2$ convolutional code, all even or odd positions in a codeword \mathbf{v} are affected only by the first or second encoding polynomial, respectively. Thus, the row distance d_j^r can be calculated as the sum of the minimum weights of all j -shifts and linear combinations of the first and second encoding polynomial, respectively. As *shifts* and *xor*-operations are only efficient up to a size of 64 bits, the sum of the memory m and the maximum row distance j_{max} is upper bounded by 64.

A.6 Row Distance

Sometimes it is necessary to calculate the first row distance values d_j^r , $j = 0, 1, 2, \dots$ for any rate $R = b/c$ convolutional code \mathcal{C} with any memory m . Therefore another, more general, implementation was developed covering any convolutional code at the expense of a higher complexity and a larger computation time. More precisely, two slightly different applications were developed:

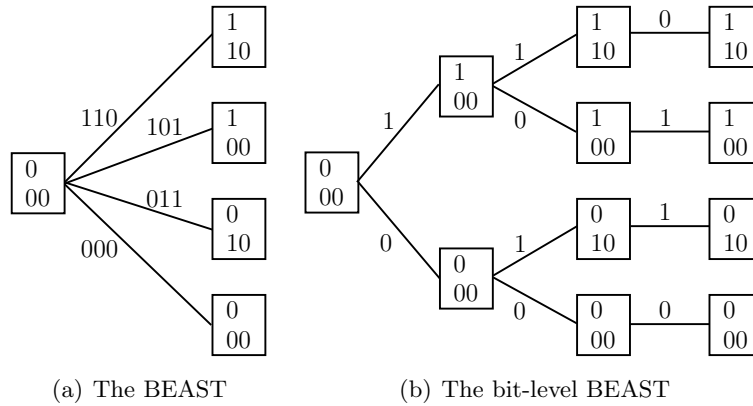


Figure A.1: Process of determining 4 children nodes for a rate $R = 2/3$ convolutional codes with encoding matrix $G = (1 + D + D^2 \quad 1 + D^2)$ by the BEAST (a) and by the bit-level BEAST (b).

one for rate $R = 1/2$ with any memory m and a second, more general, for any rate $R = b/c$ convolutional code and any memory m .

A.7 The BEAST

The BEAST — Bidirectional Efficient Algorithm for Searching code Trees — was implemented according to the description in Section 4.2. In order to reduce complexity and computation time, several limitations were introduced, leading to different implementations which are optimized according to their individual case.

As part of the software toolbox, the following implementations were considered, where m , b and c may be any integers greater than zero, and $b \leq c$:

- The BEAST for rate $R = 1/c$ convolutional code with memory m .
- The BEAST for rate $R = 1/c$ convolutional code with memory $m \leq 64$.
- The BEAST for rate $R = b/c$ convolutional code with memory $m \leq 64$.
- The BEAST for rate $R = b/c$ convolutional code with memory $m \leq 16$.
- The bit-level BEAST for rate $R = b/c$ convolutional codes with memory $m \leq 64$. In this case the process of determining the 2^b children nodes is split up into c steps: during the first b steps every node has two children, and in the remaining $c - b$ steps, every node has only one successor

(*cf.* Figure A.1(a) and A.1(b)). Notice that this implementation was developed to obtain smaller forward and backward sets.

Notice that by introducing limitations as noted above, the BEAST could be further optimized, resulting in less complex calculations, which can be performed within a shorter computation time. For example, by limiting the memory m from 64 to at most 16, the storage needed for a single vertex could be reduced by 70% for the rate $R = 5/20$ woven graph code with constituent convolutional code, mentioned in Section 7.2.