Optimized Belief-Propagation Decoding for Low-Delay Applications in Digital Communications

Optimierte iterative Decodierungsverfahren für codierte digitale Übertragung mit geringer Latenz

Der Technischen Fakultät der
Friedrich-Alexander-Universität Erlangen-Nürnberg
zur Erlangung des Grades

Doktor-Ingenieur

vorgelegt von

Thorsten Hehn

Erlangen, 2009

Als Dissertation genehmigt von
der Technischen Fakultät der
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Tag der Einreichung:	18. Dezember 2008
Tag der Promotion:	28. Mai 2009
Dekan:	Prof. Dr.-Ing. Johannes B. Huber
Berichterstatter:	**Prof. Dr.-Ing. Johannes B. Huber**
	(Friedrich-Alexander-Universität Erlangen-Nürnberg)
	Prof. Dr. Olgica Milenkovic
	(University of Illinois at Urbana-Champaign, USA)

Acknowledgments

I would like to thank Prof. Dr. Huber and Prof. Dr. Olgica Milenkovic for supervising me during my studies. They provided me with the fundamental ideas for this work and gave me support whenever needed. It is highly appreciated that Prof. Huber followed my work closely although he became dean of our faculty and that the supervision by Prof. Milenkovic never lost its intensity after we both left the University at Boulder.

This work would also not have been possible without my friend and colleague Stefan Laendner, whom I would like to thank for great discussions on the black board and a nice and friendly atmosphere in our office. The same holds for my friend Azad Ravanshid. Furthermore, I would like to thank my friend and former office mate Volker Pauli for a good time. All other colleagues at the Institute for Information Transmission provided me with help and were great partners in academic and private discussions, thanks to them. My parents and my friends supported me whenever possible during my Ph.D. studies. I also thank the impersonated character Jack Bauer who served as a role model in work ethics.

Last but not least I would like to thank my girlfriend Vera for giving me strength and love on each day of my studies.

Abstract in German (Deutschsprachige Zusammenfassung)

Um eine gewünschte Zuverlässigkeit bei der Übertragung von Nachrichten zu gewährleisten, nutzen alle modernen Kommunikationssysteme Kanalcodierungsverfahren. Dabei sind insbesondere hocheffiziente Verfahren mit mäßigem Aufwand für die Decodierung und geringer Verzögerung des Datenstroms von Interesse. Seit 1993 [BGT93] ist bekannt, dass die iterative Decodierung von Kanalcodes zu einer sehr hohen Leistungsfähigkeit bei moderater Decodierkomplexität führt. Insbesondere die Low-Density Parity-Check Codes [Gal63, MN96], welche sich durch eine dünn besetzte Prüfmatrix auszeichnen, und der iterative Belief-Propagation Decodieralgorithmus sind Gegenstand der aktuellen Forschung. Mit sehr langen Low-Density Parity-Check Codes und dem Belief-Propagation Algorithmus gelingt es, Kanalcodierung nahe an den theoretischen Grenzen [RSU01] zu betreiben. Durch lange Codes entstehen jedoch signifikante Verzögerungen im Übertragungssystem. Von diesem Nachteil sind auch die ersten Low-Density Parity-Check Codes, die in Standards [JTC05, IEE06] verwendet werden, betroffen.

Die vorliegende Arbeit geht der Frage nach, wie eine hohe Leistungsfähigkeit mit iterativen Decodierverfahren bei möglichst geringer Verzögerung erreicht werden kann. Zu diesem Zweck werden zunächst Untersuchungen an iterativen Kanalcodierungsverfahren für den binären Auslöschungskanal durchgeführt. Publikationen auf diesem Gebiet [SV06, HS07] berichten über eine höhere Leistungsfähigkeit bei der Decodierung von kurzen algebraischen Codes. Diese wird durch eine *redundante* Codebeschreibung erreicht, die dem iterativen Decoder zur Verfügung gestellt wird.

Für den binären Auslöschungskanal wird in dieser Arbeit eine neue Methode vorgestellt, die eine redundante Beschreibung im Bedarfsfall erzeugt und bisher publizierte Methoden hinsichtlich der Leistungsfähigkeit übertrifft. Theoretische Überlegungen zeigen in diesem Zusammenhang auf, dass bestimmte Arten von Codebeschreibungen für die Decodierung vorteilhaft sind. Für den binären Auslöschungkanal ist damit eine Methode gefunden, um bei gegebener Verzögerung die Decodierfähigkeit signifikant zu verbessern. Voraussetzung dafür ist die Verwendung algebraischer Codes.

Diese Resultate werden anschließend auf ein Kanalmodell, welches dem Signal additives weißes Gaußsches Rauschen überlagert, übertragen. Da bei diesem Modell dem Empfänger Daten über die Verlässlichkeit der empfangenen Werte vorliegen, können prinzipiell bessere De-

codierverfahren angewandt werden. Die Rückkopplung von nicht unabhängigen und im Kanal verfälschten Informationen im iterativen Decodierprozess führt jedoch zu Einbußen, wodurch sich der Leistungsunterschied zwischen optimalen und iterativen Verfahren begründet. Die Herausforderung bei der Anwendung von redundanten Codebeschreibungen bei diesem Kanalmodell besteht darin, dass durch diese die genannten Rückkopplungseffekte erheblich verstärkt werden. Aufgrund dieser Tatsache nimmt die Leistungsfähigkeit im Vergleich zum herkömmlichen iterativen Verfahren stark ab. Der in der Arbeit vorgestellte *Multiple-Bases Belief-Propagation* Ansatz lindert dieses Problem, indem die Redundanz auf mehrere Decoder verteilt wird und deren Ergebnisse wieder zusammengeführt werden, so dass die redundante Codebeschreibung effektiv genutzt werden kann. Verschiedene Methoden des genannten Algorithmus mit unterschiedlichen Vorzügen werden vorgestellt und anhand von kurzen, algebraischen Codes verglichen. Im Anschluß wird gezeigt, wie das vorgestellte Verfahren auf Codes aus dem 802.16e WiMAX Standard [IEE05] und Low-Density Parity-Check Codes, die mit dem Progressive Edge-Growth Algorithmus optimiert wurden, angewandt werden kann. Es ist bemerkenswert, dass das vorgestellte Verfahren bei letzterer Klasse von Kanalcodes die Leistungsfähigkeit im Vergleich zur Decodierung mit dem Belief-Propagation Algorithmus verbessern kann. Es gilt in diesem Zusammenhang zu beachten, dass diese Codes bei gegebener Länge die derzeit beste Decodierleistung auf dem Gebiet der iterativen Decodierung bieten. Damit kann auch bei Übertragung über den genannten additiven Rauschkanal die Leistungsfähigkeit der Decodierung bei gegebener Verzögerung erhöht werden.

Insgesamt lässt sich sagen, dass die vom Progressive Edge-Growth Algorithmus optimierten Codes, die in dieser Arbeit betrachtet werden, weniger als 0.50 dB von der informationstheoretischen Gallager Existenzschranke entfernt sind. Der Gewinn, der auf das beschriebene Verfahren zurückgeht, hängt von der Länge des Codes ab und liegt zwischen 0.08 dB und 0.57 dB.

Das genannte Verfahren wird abschließend mit weiteren bekannten Ansätzen verglichen. Hierzu wird die strukturbedingte Verzögerung des Datenstroms durch Kanalcodierungsverfahren untersucht und basierend auf diesem Maß ein Vergleich zwischen Block- und Faltungscodes durchgeführt. Letztere zeichnen sich durch gute Eigenschaften bei geringer Verzögerung aus. Es wird geklärt, in wie weit die modernen Low-Density Parity-Check Codes im Zusammenspiel mit redundanten Codebeschreibungen sich mit Faltungscodes in Bezug auf Leistungsfähigkeit und Verzögerung vergleichen. Es stellt sich heraus, dass die Antwort auf diese Frage von der gewünschten Anwendung abhängig ist. Während bei einer extrem kleinen Verzögerung des Datenstroms sich Faltungscodes als überlegen erweisen, erzielen moderne Codierverfahren im Zusammenspiel mit den obigen Ansätzen bereits ab mäßigen Verzögerungen deutliche Gewinne bei der Leistungseffizienz. Abschließend wird gezeigt, dass die strukturelle Verzögerung auch in praktischen Anwendungen, bei denen Verzögerungen durch Prozessorlaufzeiten entstehen, ein relevantes Maß darstellt.

Abstract

All modern communications systems apply channel coding schemes in order to guarantee the quality of service expected by the user. It is the aim of coding theorists to design schemes with high decoding performance, low-complexity decoder implementations, and a low data delay. Since 1993 [BGT93] it is known that iterative decoding of channel codes is one way to obtain good performance results with realizable decoders. Current research activities focus mainly on the class of low-density parity-check codes [Gal63, MN96] and the iterative belief-propagation decoder, usually applied to decode these codes. It was shown in [RSU01] that near-optimum transmission is possible using low-density parity-check codes of very long length and belief-propagation decoding. However, long block codes impose a high delay on the transmitted data stream and ask for more decoding complexity. This is also an issue for the low-density parity-check codes used in digital communication standards [JTC05, IEE06].

In this thesis, we investigate how a good decoding performance can be obtained by codes of short to moderate delay which are decoded by an iterative decoder. As a first step, channel coding schemes used for signaling over the binary erasure channel are considered. It was shown recently that the decoding performance of short algebraic codes, decoded by a belief-propagation decoder, can be improved significantly if the iterative decoder is provided with a *redundant* code description [SV06, HS07].

We introduce a novel decoding method for this type of channel. This approach creates the required redundancy on request and outperforms the schemes currently known in literature. Furthermore, we deduct by theoretical considerations that a certain kind of code description is most suitable for decoding these codes in an iterative manner. With this, a method to improve the decoding performance on the binary erasure channel for given delay is found. However, algebraic codes are needed to apply the method.

A second step transfers the results for the binary erasure channel to a channel model which distorts the transmitted data with additive white Gaussian noise. In this case, the received data allows to calculate reliability values for each position in a codeword. Consequently, more sophisticated iterative decoders can be applied. However, these schemes are not optimal, what is due to the feedback of non-independent and possibly distorted data within this process. This poses a challenge for the use of redundant code descriptions, as the redundancy intensifies the feedback process. Hence, the straightforward application of redundant code descriptions leads to impaired performance results when compared to usual iterative decoding. The *multiple-bases*

belief-propagation approach, to be introduced in this thesis, mitigates this problem, as it partitions a redundant code description to a multitude of belief-propagation decoders in such a way that each decoder observes no or only little redundancy. This approach, together with the ability to merge the results of the different decoders, allows to use the redundant code description efficiently and to obtain performance improvements. Several variations of this approach are introduced and their advantages and disadvantages are shown using algebraic codes of short length. Subsequently, the field of application is extended to codes from the IEEE 802.16e WiMAX standard [IEE05], as well as codes optimized by the progressive edge-growth algorithm. Especially the latter codes are of interest, as they currently provide the best performance on an iterative decoder when rate and length of the code are given. It is a remarkable result that the performance of these codes can be improved further by the proposed scheme. Consequently, a way is found to improve the decoding performance at a given delay when transmitting over the additive white Gaussian noise channel.

Currently, the codes optimized by the progressive edge-growth algorithm considered in this work show a gap of at most 0.50 dB to the Gallager bound, a well-known existence bound from the field of information theory. Included in this result is a code-length dependent gain of 0.08 dB to 0.57 dB, which is obtained by the multiple-bases belief-propagation approach.

As a third step, we assess our results in a more general manner and compare them to further schemes from channel coding. To this end, we introduce the structural delay, which is an indicator of the delay solely imposed by the channel coding scheme. The structural delay allows to compare the proposed approach to schemes which are not block-oriented, such as convolutional codes, while keeping the focus on both the decoding performance and the imposed data latency. As convolutional codes are known to yield good performance results at low data latency, it is of great interest to relate them to the modern low-density parity-check codes using the introduced decoding approaches. A general recommendation in favor of one class of codes cannot be given, but application-dependent recommendations can be made. If a very short delay is targeted, convolutional codes are clearly still ahead of their block-oriented counterpart. However, starting from a moderate delay, block codes using the proposed scheme significantly outperform convolutional codes in terms of decoding performance. As a final step, it is shown that the structural delay is a valid measure for the overall delay which includes the latency introduced by limited computational power.

Contents

Chapter 1

Introduction and Outline

A lot of communication systems were successfully employed during the last decade, many of them allowing consumers to transmit and receive data with their own equipment. Due to the pricing of bandwidth and because of regulations, these systems need to be highly efficient and have to provide the expected quality while transmitting with low power. The application of powerful channel codes is an indispensable technique for transmitting data in a reliable way and with high efficiency.

The fundamentals of channel coding were already established in [Sha48], where the channel coding theorem was proposed and proven. Since then, it is well-known that reliable communication can be achieved by using a sufficiently long channel code together with an optimal decoder. Many coding schemes were proposed, e.g. by [Gol49, BRC60], and all of them aimed at maximizing the (minimum) Hamming distance between codewords. It took until 1993 [BGT93] to discover that iterative decoding methods are one possibility to decode very long codes with reasonable complexity. This method allows for good decoding performance although the iterative decoder is a suboptimal decoder and the codes usually applied show poor distance profiles. A first class of codes used in iterative schemes are convolutional codes concatenated by random interleavers (turbo codes, [BGT93]). Three years after the introduction of turbo codes, it was reported that iterative decoding also leads to desirable results with a special class of block codes [MN96]. These codes are known as low-density parity-check (LDPC) codes and were originally discovered in [Gal63], but ignored until 1996. In a groundbreaking paper published in 2001 [RSU01], Richardson, Shokrollahi, and Urbanke derived the properties of parity-check matrices of very long codes which approach the capacity very closely. This insight allowed to design long codes with good decoding performance. In this context, it was assumed that a suboptimal iterative decoder is used for the decoding process. Subsuming, decoding of very long codes with iterative decoders is possible and leads to near-optimum performance if proper parity-check matrices are chosen.

However, in all delay-critical (two-way) communication systems, long channel codes cannot be used as they cause high data latency. For short codes, the communication scenario is different from the one described above. The decoding performance of LDPC codes with iterative decoders

leads to suboptimal results, especially when the design rules from [RSU01] for infinite-length codes are followed. In this thesis, we are concerned with the design of good iterative channel coding schemes with low delay. We show methods to improve the decoding performance of existing block codes while keeping all code parameters and the delay constant. We restrict our investigations to block codes of length up to $n = 1000$. We also conduct a comparative performance study with convolutional codes.

As this thesis initiates the studies on LDPC codes at the Institute for Information Transmission (University of Erlangen-Nuremberg), a detailed background part (Chapters 2 to 5) is included. These chapters mark a good starting point for succeeding students and allow readers already familiar with the subject to follow the exposition in the second part with ease.

The latter part (Chapters 6 to 8) contains the novel contributions of this thesis. We first provide a decoding method for the binary erasure channel generating redundant parity information on demand. This method is suitable for the class of cyclic codes and allows for significant performance improvements compared to standard techniques. Theoretical investigations are conducted to identify the advantages of parity-check matrices of cyclic form for iterative decoding. These results inspire a redundant decoding method applicable to the additive white Gaussian noise channel, which is the second contribution of this work. We propose several variations of this method and show its applicability by testing it on various codes, such as optimized LDPC codes and codes used in an communications standard by the Institute of Electrical and Electronics Engineers (IEEE). As a third contribution, we compare several convolutional codes and block codes with respect to the delay imposed by channel coding (structural delay) and their decoding performance. Convolutional codes are known to achieve good performance results while causing a data latency which is comparable to that of short block codes. Hence, it is of great interest to compare their performance to modern optimized codes, with and without making use of the novel methods described above. As a final result, we present an overview of current coding schemes with respect to delay and decoding performance and relate these results to well-known bounds from information theory.

Chapter 2

Channel Models and Transmission Setup

We introduce the channel models considered in this thesis and provide a detailed description of the transmission scheme. For a rigorous presentation, we define the variables in the transmission setup.

The channel is defined as the part of the communications system that has to be accepted by the developer as it is. The developer cannot change it or is not willing to do this. This reasonable definition goes back to John L. Kelly [Hub07]. Channels include both *transmission channels*, such as e.g. telephone lines and satellite links as well as *storage media* like magnetic and optical data devices. The channel models used for the investigations in this thesis are the binary erasure channel (BEC) and the additive white Gaussian noise (AWGN) channel. Channel models are used to design and analyze transmission systems, and are furthermore used in simulation setups. The BEC model is used to replicate a transmission with packet loss, while the AWGN channel model is a simple and general model for noisy transmission schemes. The Gaussian noise originates from the uncorrelated thermal noise processes of the assembly units in the transmitter and the receiver, but also from the superposition of uncorrelated noise sources that affect the signal on the transmission path. The AWGN channel is widely used to model communication over cables or when signaling over the air. This scheme is often extended by models for signal interference and signal fading.

Often channel models are concatenated to mimic the behavior of a real-world channel. For example, the satellite communication channel is modeled as the concatenation of an AWGN channel and a BEC [DHLH06]. For storage applications, the partial response channel [Hol91, II97] is used to emulate the channel. This model includes a linear filter and the AWGN channel. In this thesis, we restrict our attention to the fundamental BEC and AWGN channel model.

2.1 Binary Erasure Channel

The BEC channel was introduced by Peter Elias (cf. e.g. [RU08]), who used it in 1954 as the simplest example of a channel. Despite its simplicity, studies on this channel are important for two reasons. First, the simple channel model often inspires approaches that can be (partially) transferred to more complicated models. In fact, one approach for the AWGN channel, which is considered in this thesis, is based on investigations performed on the BEC. Second, the popularity of the internet with its packet routing strategies has lifted the BEC into the world of applications. The loss of a packet on a specific route corresponds to an erasure. The BEC and channel coding schemes used to signal over this channel are still topics of active research. Recently, several researchers [SV06, HT07, HS07] proposed novel schemes for iterative decoding of data signaled over the BEC.

Let us now consider this channel model in more detail. The BEC either transmits a binary input symbol correctly or erases it. In the latter case, we denote the received symbol by '?'. The probability for an erasure is denoted by ε and is independent of the input symbol, i.e.

$$\Pr(Y_i = ? \mid X_i = 0) = \Pr(Y_i = ? \mid X_i = 1) = \varepsilon.$$

In this context, X denotes a discrete memoryless source that corresponds to a sequence of independently, identically distributed (IID) random variables (RV). We write $X_i = x_i$ to denote the event that the RV X_i takes on the value $x_i \in \mathcal{X}$ at time step i, $i \in \mathbb{Z}$, where $\mathbb{Z}$ denotes the set of integers and $\mathcal{X}$ denotes a binary alphabet. Figure 2.1 depicts a binary erasure channel. Note that the erasures introduced by this channel do not cause loss of synchronization at the

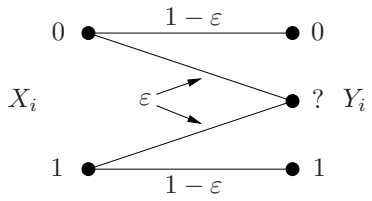


Figure 2.1: Binary erasure channel with erasure probability ε.

receiver. Instead, the exact positions of the erased symbols are known.

The capacity of the BEC is straightforward to calculate as it can be decomposed in strongly symmetric channels [Hub07, p. 160] with capacities $C = 0\,\frac{\text{bit}}{\text{use}}$ and $C = 1\,\frac{\text{bit}}{\text{use}}$ and subchannel probabilities ε and $1-\varepsilon$, respectively. The resulting overall capacity is achieved for equiprobable input symbols and reads

$$C_{\text{BEC}} = 1 - \varepsilon\,\frac{\text{bit}}{\text{use}}. \tag{2.1}$$

2.2 AWGN Channel

The second channel model discussed in this thesis is the AWGN channel. We consider digital pulse amplitude modulation (PAM) at carrier frequencies $f_\mathrm{c} > 0$. First, we focus our attention on a continuous-time model. In this context, we denote the continuous time by t. Based on this model, we introduce the AWGN channel model in discrete time, which is used throughout this thesis. In the following, we deploy the equivalent complex baseband (ECB) representation [Hub06, Tre01] with respect to f_c. We define this transformation by means of the continuous-time transmission signal $s_\mathrm{RF}(t)$. The transformation to its representation in the ECB domain with respect to f_c reads

$$s(t) = \frac{1}{\sqrt{2}} \left(s_\mathrm{RF}(t) + \mathrm{j}\mathcal{H}\{s_\mathrm{RF}(t)\} \right) \cdot \mathrm{e}^{-\mathrm{j}2\pi f_\mathrm{c} t},$$

where $\mathcal{H}\{s_\mathrm{RF}(t)\} = s_\mathrm{RF}(t) \star \frac{1}{\pi t}$ is the Hilbert transform of the signal $s_\mathrm{RF}(t)$, the operator $\star$ denotes the convolution, j is the imaginary unit with $\mathrm{j}^2 = -1$, and e is the Euler number. This representation transforms radio-frequent signals into the baseband. The corresponding inverse transformation is given by $s_\mathrm{RF}(t) = \sqrt{2}\mathrm{Re}\{s(t) \cdot \mathrm{e}^{\mathrm{j}2\pi f_\mathrm{c} t}\}$. The operator $\mathrm{Re}\{\cdot\}$ denotes the real part of a complex variable.

The resulting model of a transmission over the AWGN channel for digital PAM transmission is given in Figure 2.2. Note that all variables except $x_{i'}$ are in general taken from the set of complex numbers, denoted by $\mathbb{C}$. Besides the complex variables, this model includes a mapper, a

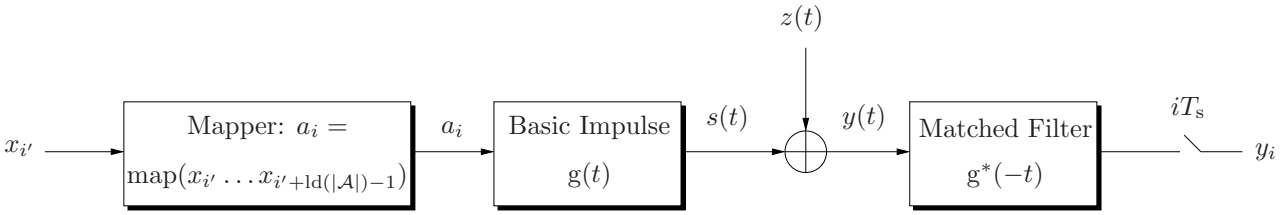


Figure 2.2: AWGN channel model (including devices necessary for digital PAM transmission). All time processes are given in the ECB domain.

modulator, and a matched filter. Usually, the binary data is not signaled directly over the AWGN channel. Instead, mapped symbols are transmitted. The mapper assigns encoded data vectors to signal points from a given signal constellation $\mathcal{A}$. The signal point a_i is generated according to the mapping function $a_i = \mathrm{map}(x_{i'}, \ldots x_{i'+\mathrm{ld}(|A|)-1})$ with $i, i' \in \mathbb{Z}$. Here, the operator $|\cdot|$ denotes the cardinality of a set, and a_i represents a realization of the RV A_i at time step i. The PAM signal constellation specifies the PAM subtype of the digital modulation scheme like amplitude-shift keying (ASK), where the elements of $\mathcal{A}$ are real, or phase-shift keying (PSK), where the elements of $\mathcal{A}$ are uniformly spaced on the unit circle in the complex plane, and quadrature-amplitude modulation (QAM), where the elements of $\mathcal{A}$ are taken from Gaussian integers [Wei02] with odd

real and imaginary part. We point out that in the case of $|\mathcal{A}| = 2$, the differentiation between the indices i' and i is not required and the mapping function simplifies to $a_i = \mathrm{map}(x_i)$.

In order to create a continuous-time signal, the current signal point is modulated to a PAM signal by means of a basic impulse. This impulse is denoted by $\mathrm{g}(t) \in \mathbb{C}$ and its duration, referred to as the symbol interval, equals T_s. The transmitted time-continuous signal reads

$$s(t) = \sum_{i=-\infty}^{\infty} a_i \cdot \mathrm{g}(t - i \cdot T_\mathrm{s}).$$

Note that T_s specifies the time to transmit a (mapped) transmission symbol. If BPSK is used, it also specifies the time to transmit an encoded symbol. Additive noise, henceforth denoted by $z(t)$, superimposes the transmitted signal. The resulting signal, present at the receiver input, is denoted by $y(t)$, cf. Figure 2.2. The additive noise $z(t)$ is a stationary complex Gaussian random process with a constant power-density spectrum $\Phi_{zz}(f) = N_0$. In this context, we refer to N_0 as the noise-power spectral density. The real part and the imaginary part of $z(t)$ are independent of each other. Moreover, noise realizations at different time steps t_1 and t_2 are independent of each other. As a consequence, $\Phi_{\mathrm{Re}\{z\}\mathrm{Re}\{z\}}(f) = \Phi_{\mathrm{Im}\{z\}\mathrm{Im}\{z\}}(f) = N_0/2$, i.e. the real part and the imaginary part of the noise process contribute to its power to the same degree. At the receiver, a matched filter $\mathrm{g}^*(-t)$ and a sampling device are used to obtain the discrete-time sequence y_i. Here, the operator $(\cdot)^*$ denotes the conjugated complex of a complex variable. The basic impulse and the corresponding matched filter are chosen such that no interference between two consecutive transmissions exists, i.e. such that they fulfill an orthogonality condition, given by

$$\frac{1}{T_\mathrm{s}} \int_{-\infty}^{\infty} \mathrm{g}(t + jT_\mathrm{s}) \cdot \mathrm{g}^*(t)\,\mathrm{d}t = \begin{cases} 1 & \text{for} \quad j = 0 \\ 0 & \text{for} \quad j \neq 0 \end{cases} . \tag{2.2}$$

This criterion is known as the Nyquist criterion [Pro08, p. 605]. We henceforth assume that this criterion, which specifies the basic impulse $\mathrm{g}(t)$ as a square-root Nyquist impulse [Pro08, p. 605], is fulfilled. The autocorrelation $\mathrm{g}(t) \star \mathrm{g}^*(-t)$ of any square-root Nyquist impulse $\mathrm{g}(t)$ is a Nyquist impulse, cf. Equation (2.2).

As the orthogonality condition is fulfilled, there is no interference between subsequent transmission symbols. Hence, a non-dispersive time-discrete model can be used. The noise samples at the output of the matched filter, henceforth modeled by z_i, are uncorrelated and statistically independent as AWGN noise is added in the continuous-time domain. Note that we are using the ECB representation and hence all signals are complex variables. Figure 2.3 shows a block diagram of a time-discrete AWGN channel where we assume that the additive white Gaussian noise has variance σ_z^2. This variance equals the power of the noise signal. The variable $z_i \in \mathbb{C}$ is modeled as a realization of the complex RV Z_i, and denotes the additive discrete-time complex white Gaussian noise in the ECB. Its probability density function (PDF) is centered around the origin of the complex plane and equals

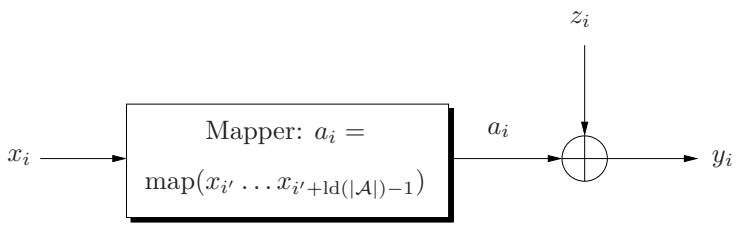


Figure 2.3: Binary input AWGN channel at time step i.

$$\mathrm{PDF}_{Z_i}(z_i) = \frac{1}{2\pi\sigma_\mathrm{R}\sigma_\mathrm{I}} \mathrm{e}^{-\left(\frac{\mathrm{Re}\{z_i\}^2}{2\sigma_\mathrm{R}^2}+\frac{\mathrm{Im}\{z_i\}^2}{2\sigma_\mathrm{I}^2}\right)}, \forall i, \tag{2.3}$$

where $\sigma_\mathrm{z}^2 = \sigma_\mathrm{R}^2 + \sigma_\mathrm{I}^2$ holds. There, σ_R and σ_I are the variances of $\mathrm{Re}\{Z_i\}$ and $\mathrm{Im}\{Z_i\}$, respectively. Due to the rotational invariance of the stationary complex process [Gre65], $\sigma_\mathrm{R} = \sigma_\mathrm{I}$ holds. The operator $\mathrm{Im}\{\cdot\}$ denotes the imaginary part of a complex variable.

The PDF of the channel output is given by

$$\mathrm{PDF}_Y(y) = \sum_{a_i \in \mathcal{A}} \Pr(A_i = a_i) \mathrm{PDF}_{Z_i}(y - a_i).$$

Figure 2.4 shows the PDF of the received RV Y for quadrature phase-shift keying (QPSK). We show two projections, where the two constraints $\mathrm{Im}\{y\} = 1$ and $\mathrm{Re}\{y\} = 1$, respectively, are set.

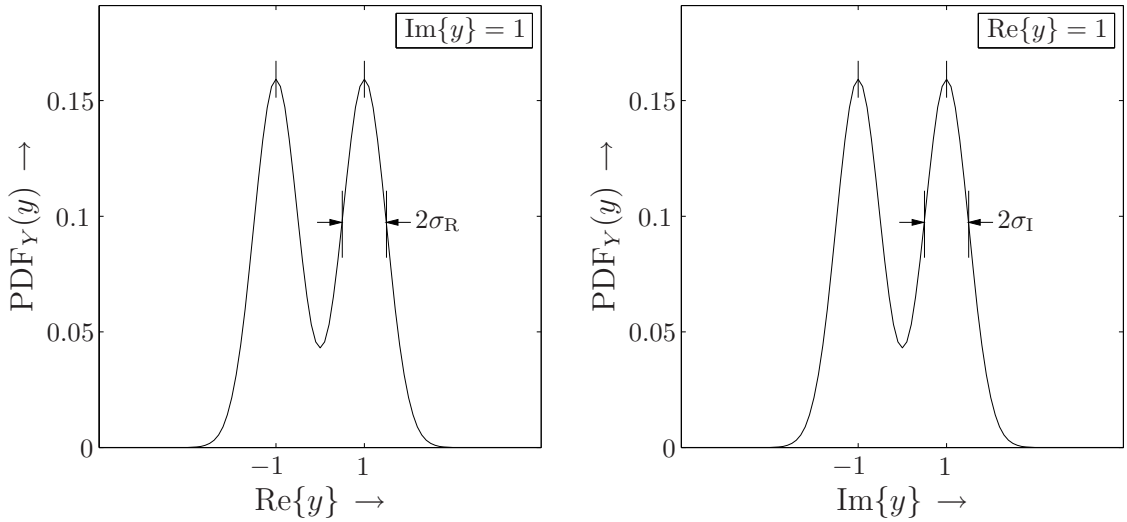


Figure 2.4: Two projections of the probability density functions of Y for $\mathcal{A} = \{-1 - \mathrm{j}, -1 + \mathrm{j}, +1 - \mathrm{j}, +1 + \mathrm{j}$ and $\sigma_\mathrm{R} = \sigma_\mathrm{I} = 0.5$. QPSK is the signal constellation used in this coherence.

In order to connect the variance σ_z^2 of the time-discrete noise to the noise-power spectral density N_0, we revisit the continuous-time channel model. Figure 2.5 shows the power-density spectrum for the real and the imaginary part of the noise process. Let us now calculate the power of the noise $z(t)$. The AWGN channel model may seem to be impractical as one needs to

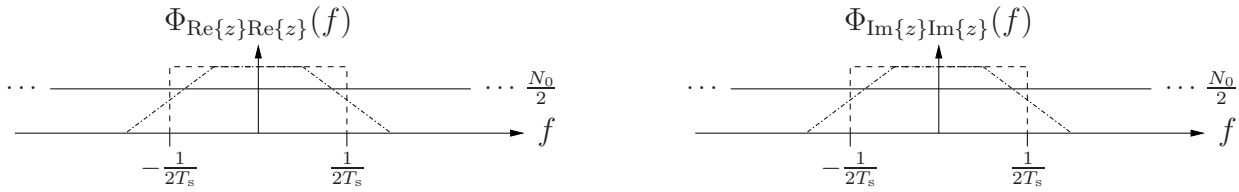


Figure 2.5: Complex white noise and the power transfer functions of two receiver input filters. Each dimension contributes to the constant noise-power spectral density with $N_0/2$.

calculate the integral of the power-density spectrum $\Phi_{zz}(f)$ over all $f \in \mathbb{R}$ for each dimension, what results in unlimited power. Here, $\mathbb{R}$ denotes the set of real numbers. However, one restricts the receiver input to the frequency range data is being transmitted on. When using PAM, this frequency range depends only on the symbol interval T_s. In the ECB domain, these frequencies are centered around $f = 0\,\mathrm{Hz}$ and the range is limited to $-(1+\alpha_\mathrm{bw})\frac{1}{2T_\mathrm{s}} \leq f \leq \frac{1}{2T_\mathrm{s}}(1+\alpha_\mathrm{bw})$ [Hub06]. The variable α_bw is the bandwidth extension factor. The dashed and the dash-dotted lines in Figure 2.5 show the power transfer functions of two receive filters which satisfy the Nyquist criterion in the frequency domain, i.e. the corresponding basic impulses of these filters fulfill Equation (2.2). The dashed filter is an academic example as it represents an idealized receive filter. The dash-dotted filter has a bandwidth extension factor of $\alpha_\mathrm{bw} = 0.5$ and is hence more practical. They restrict the receiver input to base-band frequencies of range $1/(T_\mathrm{s})$ and $1.5/(T_\mathrm{s})$, respectively. Both filters allow for transmission at a data rate of $1/T_\mathrm{s}$ without intersymbol interference and limit the power of the noise to a finite value which is equal for both. These values correspond directly to the variance of the time-discrete noise process if the energy of the basic impulse $\mathrm{g}(t)$ is normalized to 1 [Hub06, p. 497].

We calculate the noise power of the time-discrete signal as

$$\sigma_\mathrm{z}^2 = \sigma_\mathrm{R}^2 + \sigma_\mathrm{I}^2 = \frac{N_0}{2T_\mathrm{s}} + \frac{N_0}{2T_\mathrm{s}} = \frac{N_0}{T_\mathrm{s}}.$$

If the ECB signal is a real signal, i.e. if one transmits an ASK signal, only the real part of the noise with power $\sigma_\mathrm{R}^2 = \frac{N_0}{2T_\mathrm{s}}$ is relevant in the receiver. Note that all variances are expressed in the unit Watt (W). Using the coherence $E_\mathrm{s} = S \cdot T_\mathrm{s}$, where S denotes the signal power and E_s is the energy per symbol, one obtains

$$\frac{E_\mathrm{s}}{N_0} = \frac{S \cdot T_\mathrm{s}}{N_0} = \frac{S}{\sigma_\mathrm{z}^2}.$$

Setting $S = 1\,\mathrm{W}$ leads to the well-known formulas

$$\frac{E_\mathrm{s}}{N_0} = \frac{1}{\sigma_\mathrm{z}^2/\mathrm{W}}$$

for the complex case and

$$\frac{E_\mathrm{s}}{N_0} = \frac{1}{2\sigma_\mathrm{R}^2/\mathrm{W}}$$

for the real-valued case. The capacity of the time-discrete real and complex AWGN channel reads $C_{\mathrm{AWGN}} = \frac{1}{2}\mathrm{ld}(1 + S/\sigma_{\mathrm{R}}^2)\,\frac{\mathrm{bit}}{\mathrm{use}}$ and $C_{\mathrm{AWGN}} = \mathrm{ld}(1 + S/\sigma_{\mathrm{z}}^2)\,\frac{\mathrm{bit}}{\mathrm{use}}$ [Sha48], respectively, where $\mathrm{ld}(\cdot)$ denotes the logarithm with base 2. The capacity is achieved for continuous channel input variables with Gaussian PDF.

For simplicity, we restrict our attention to binary phase-shift keying (BPSK) for signaling over the AWGN channel. Consequently, we restrict the mapping function to the BPSK mapping function

$$\mathrm{map}(x_i) = \begin{cases} a_i = 1 & \text{if} \quad x_i = 0 \\ a_i = -1 & \text{if} \quad x_i = 1 \end{cases} .$$

If Gray-Mapping [LC04] is applied, QPSK modulation can be understood as two independent BPSK modulations in the real and imaginary dimension. The orthogonality condition, cf. Equation (2.2), separately holds for the real and imaginary dimension. Hence, all error rates presented in this thesis directly hold for QPSK modulation, too.

Remark 2.2.1 (Forward error correction) *We point out that the correct identification for the process of error control coding (ECC), usually referred to as forward error correction (FEC), depends on the channel model. The binary erasure channel does not introduce errors but erasures, therefore the denomination "erasure recovery" is more appropriate. Also, the signal y received from the AWGN channel does not contain "hard errors". These errors would result from an immediate quantization of the signal. For this reason, "error prevention" is the first choice for a proper indication of the decoding process.*

2.3 Transmission Setup

This section describes the complete transmission setup, which includes the proposed channel models and the used channel coding schemes. To this end, we introduce T_{b} as the duration for one bit [Hub07], i.e. T_{b} is the time required to transmit one binary symbol from a non-redundant source. It is henceforth assumed that the source is free of redundancy. For a rigorous presentation we first define a channel code $\mathcal{C}$.

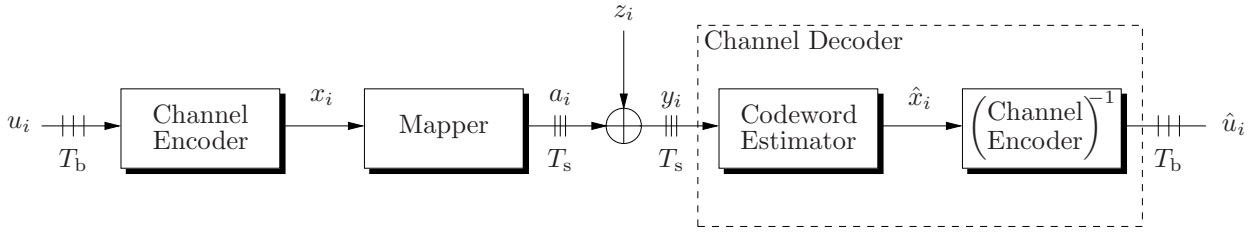


Figure 2.6: Transmission setup for signaling over the AWGN channel.

Definition 2.3.1 *The set $\mathcal{C}$ of codewords which are images of at least one source word is called the code.*

In a first step, we consider the case that a linear $[n, k, d]$ block code is used, where n, k, and d denote the length, dimension, and minimum distance of the code, respectively. Figure 2.6 depicts the (time-discrete) transmission setup. The channel encoder transforms blocks of source symbols $\boldsymbol{u} := [u_1, \ldots, u_k]$ of length k to blocks of encoded symbols $\boldsymbol{x} := [x_1, \ldots, x_n]$ of length n, u_i, $x_i \in \mathcal{X}\ \forall i$. The variable u_i is the realization of the information source and a realization of the RV U_i. The rate of a binary channel code is given by the average information per codesymbol, i.e. $R_\mathrm{c} = \frac{\mathrm{ld}(|\mathcal{C}|)}{n} \frac{\text{bit}}{\text{codesymbol}}$, where we assume that all codewords are equally probable. If we further assume that the source words carry exactly k bits of information, and that a one-to-one mapping of source words to codewords exists, we obtain the well-known result $R_\mathrm{c} = \frac{k}{n} \frac{\text{bit}}{\text{codesymbol}}$. The encoding step is followed by the mapping to transmission symbols $\boldsymbol{a} = [a_1, \ldots, a_{n'}]$, $a_i \in \mathcal{A}$ with $\frac{n'}{n} = \frac{\mathrm{ld}(|\mathcal{X}|)}{\mathrm{ld}(|\mathcal{A}|)}$. The resulting overall rate equals $R = \frac{k}{n'} \frac{\text{bit}}{\text{channel use}}$. Throughout this thesis, we assume $|\mathcal{A}| = |\mathcal{X}| = 2$ and thus $R = R_\mathrm{c}$.

The channel decoder receives the vector $\boldsymbol{y} := [y_1, \ldots, y_n]$ and finds an estimate for the transmitted codeword, denoted by $\hat{\boldsymbol{x}} := [\hat{x}_1, \ldots, \hat{x}_n]$. The corresponding source symbols are found by inverting the encoding process, and are denoted by $\hat{\boldsymbol{u}} := [\hat{u}_1, \ldots, \hat{u}_k]$. Again, we consider y_i, $\hat{x}_i$ and $\hat{u}_i$ as realizations of the RVs Y_i, $\hat{X}_i$ and $\hat{U}_i$, respectively. Note that the extraction of the source symbols has an impact on the residual error rate, which is measured between the information source and sink. We consider this topic in detail in Chapter 5.

In order to make our results comparable to transmission systems of different rates, we introduce the signal-to-noise ratio which includes a normalization with respect to the rate. It involves the energy required to transmit one bit of information (one binary source symbol) instead of an encoded symbol and leads to the normalized signal-to-noise ratio

$$\frac{E_\mathrm{b}}{N_0} = \frac{E_\mathrm{s}}{N_0} \cdot R,$$

where E_b denotes the energy for transmission of one bit of information.

If the data is signaled over the BEC, a slightly different transmission setup is obtained, shown in Figure 2.7.

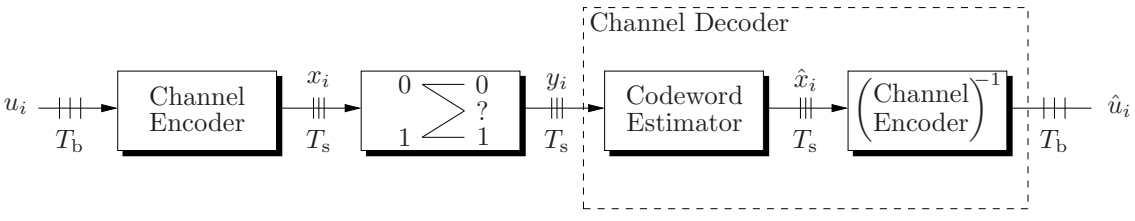


Figure 2.7: Transmission setup for signaling over the BEC channel.

We point out that the introduced transmission setups can be transferred directly to the case of transmission with convolutional codes. The transmission setups shown in Figure 2.6 and Figure 2.7 as well as the durations T_b and T_s are valid for this scenario. The encoding of convolutional codes is stream oriented (details follow in Chapter 5). We deploy only convolutional codes of rate $R = 1/n$ in this thesis. Unlike block codes, these codes do not obtain good performance from long blocks, but from encoder memory [LC04]. Assuming that n denotes the number of encoded symbols initiated by k source symbols, the rate can be calculated by $R = \frac{k}{n}$. We state that it has to be assumed that no *puncturing* [JZ99] is used, what is the case in this thesis.

Chapter 3

Basic Mathematical Operations for Iterative Decoding

Iterative decoders process multiple channel observations in order to provide an estimate for an encoded symbol. In order to describe the belief-propagation algorithm for decoding of LDPC codes rigorously, we discuss the calculation of probabilities which are conditioned on multiple observations. Note that these probabilities are of interest in several further fields, e.g. in risk analysis [CW99]. Calculating "combined" probabilities also has a long history. Research in this direction was already done by Pierre-Simon Laplace [Bac79], who especially considered the linear opinion pool, i.e. linear averaging of probabilities [GR99, CW99] as one option for calculating a "combined" probability. In this chapter, we consider probabilities of this type as well as probabilities of modulo-2 sums and show how they are calculated in an optimal way if certain assumptions hold.

3.1 Multiple Observations of an Event – Combining of Probabilities

Assume that a binary source X emits symbols from the alphabet $\mathcal{X} = \{0, 1\}$ with equal probabilities $\Pr(X = 0) = \Pr(X = 1) = 1/2$. The variable X usually denotes an encoded signal in our notation. The reader may imagine a source that directly emits encoded symbols. The output of the source cannot be observed directly, but through δ_1 different channels with output RVs Y_1 to Y_{δ_1}, cf. Figure 3.1. In this thesis we assume that these channels are binary input symmetric memoryless channels (BISMC)s.

One is interested in the probability that the event $X = x$, $x \in \{0, 1\}$ has occurred. Without loss of generality, we consider only the probability $\Pr(X = 0 \mid \text{observation})$ which provides complete information on a binary variable. In this context, the variable "observation" represents a possibly infinite set of channel output variables. The most revealing probability of the event $X = 0$ is the probability which is conditioned on all observations of the events $Y_i = y_i$ which

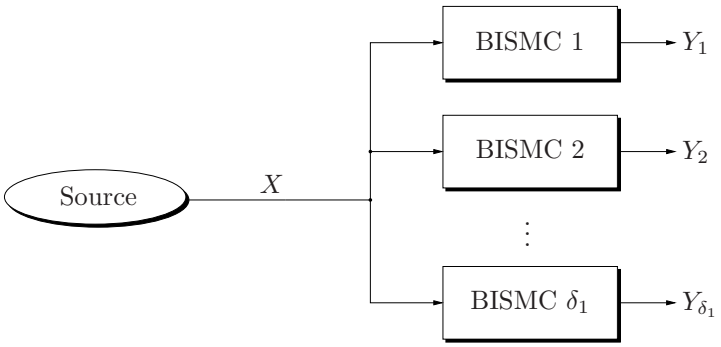


Figure 3.1: A source X can only be observed through δ_1 channels, but not directly.

are connected to event $X = 0$, i.e. the probability $\Pr(X = 0 \mid Y_1 = y_1, Y_2 = y_2, \ldots, Y_{\delta_1} = y_{\delta_1})$. Let us express this conditional probability for $\delta_1 = 2$ as a function of $\Pr(X = 0 \mid Y_1 = y_1)$ and $\Pr(X = 0 \mid Y_2 = y_2)$. The variables Y_1 to Y_2 are strongly dependent, but independently distorted. If the realization of RV X is known, the novelty of the variables Y_1 and Y_2 reduce to the added distortion and hence they can be considered as independent under the given assumption. In order to determine the conditional probability, we start with

$$p_{12} := \Pr(X = 0 \mid Y_1 = y_1, Y_2 = y_2) = \frac{\Pr(X = 0, Y_1 = y_1, Y_2 = y_2)}{\Pr(Y_1 = y_1, Y_2 = y_2)}. \tag{3.1}$$

Then,

$$\begin{aligned} p_{12} &= \\ &= \frac{\Pr(Y_1 = y_1, Y_2 = y_2, X = 0)}{\Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 0)\Pr(X = 0) + \Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 1)\Pr(X = 1)} \\ &= \frac{\Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 0)\Pr(X = 0)}{\Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 0)\Pr(X = 0) + \Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 1)\Pr(X = 1)}. \end{aligned} \tag{3.2}$$

We assume equiprobable input symbols, i.e. $\Pr(X = 0) = \Pr(X = 1) = 1/2$, and obtain

$$p_{12} = \frac{\Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 0)}{\Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 0) + \Pr(Y_1 = y_1, Y_2 = y_2 \mid X = 1)}. \tag{3.3}$$

Taking into account that Y_1 and Y_2 are independent RVs for given X, one can show that

$$p_{12} = \frac{\Pr(Y_1=y_1|X=0)\Pr(Y_2=y_2|X=0)}{\Pr(Y_1=y_1|X=0)\Pr(Y_2=y_2|X=0)+\Pr(Y_1=y_1|X=1)\Pr(Y_2=y_2|X=1)}. \tag{3.4}$$

We still need to rewrite $\Pr(Y_1 = y_1 \mid X = 1)$ and $\Pr(Y_2 = y_2 \mid X = 1)$ in terms of $\Pr(Y_1 = y_1 \mid X = 0)$ and $\Pr(Y_2 = y_2 \mid X = 0)$. It is straightforward to see that

$$\frac{\Pr(Y_1=y_1|X=1)}{\Pr(Y_1=y_1|X=0)+\Pr(Y_1=y_1|X=1)} = 1 - \frac{\Pr(Y_1=y_1|X=0)}{\Pr(Y_1=y_1|X=0)+\Pr(Y_1=y_1|X=1)}. \tag{3.5}$$

It is also known from Bayes' rule that for equal input symbol probabilities it holds that

$$\Pr(X=0 \mid Y_1=y_1) = \frac{\Pr(Y_1=y_1|X=0)}{\Pr(Y_1=y_1|X=0)+\Pr(Y_1=y_1|X=1)}. \tag{3.6}$$

Thus, Equation (3.5) can be rewritten as

$$\frac{\Pr(Y_1=y_1 \mid X=1)}{\Pr(Y_1=y_1 \mid X=0)+\Pr(Y_1=y_1 \mid X=1)} \;=\; 1-\Pr(X=0 \mid Y_1=y_1). \tag{3.7}$$

We continue by multiplying Equation (3.4) with

$$1 = \frac{\frac{1}{D_{y_1}\cdot D_{y_2}}}{\frac{1}{D_{y_1}\cdot D_{y_2}}},$$

where the denominators are set to

$$\begin{aligned} D_{y_1} &= \left(\Pr(Y_1=y_1 \mid X=0)+\Pr(Y_1=y_1 \mid X=1)\right), \\ D_{y_2} &= \left(\Pr(Y_2=y_2 \mid X=0)+\Pr(Y_2=y_2 \mid X=1)\right). \end{aligned} \tag{3.8}$$

We then obtain

$$p_{12} = \frac{\frac{\Pr(Y_1=y_1|X=0)}{D_{y_1}}\frac{\Pr(Y_2=y_2|X=0)}{D_{y_2}}}{\frac{\Pr(Y_1=y_1|X=0)}{D_{y_1}}\frac{\Pr(Y_2=y_2|X=0)}{D_{y_2}}+\frac{\Pr(Y_1=y_1|X=1)}{D_{y_1}}\frac{\Pr(Y_2=y_2|X=1)}{D_{y_2}}} \tag{3.9}$$

and finally conclude with Equation (3.5) and Equation (3.7) that

$$p_{12} = \frac{\Pr(X=0|Y_1=y_1)\Pr(X=0|Y_2=y_2)}{\Pr(X=0|Y_1=y_1)\Pr(X=0|Y_2=y_2)+(1-\Pr(X=0|Y_1=y_1))(1-\Pr(X=0|Y_2=y_2))}. \tag{3.10}$$

This formula is known as normalized multiplication [LH06] and usually given in the mnemonic form

$$p_{12} = p_1 \otimes p_2 = \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} \tag{3.11}$$

where the notation

$$p_i = \Pr(X=0 \mid Y_i=y_i),\ i \in \{1,\ldots,\delta_1\} \tag{3.12}$$

is used. Note that this formula is already given in [Gal63]. This combining of probabilities is also referred to as "parallel" information combining [LH06]. We prefer the denominator normalized multiplication as this choice avoids ambiguities with the process of combining of average mutual information. At this point, the interested reader is referred to [LH06, Ch. 4].

Let us now discuss the case $\delta_1 > 2$. It is easy to see that the calculation of a conditional probability based on δ_1 observations can be done recursively and in a "local" manner. One first concentrates on two observations and calculates the optimal probability $p_{12} = \Pr(X = 0 \mid Y_1 = y_1, Y_2 = y_2)$ by means of the normalized multiplication. If a third independent observation $Y_3 = y_3$ is available, the normalized multiplication of the hypersymbol y_1y_2 and the symbol y_3 is conducted, i.e. $p_{123} = \Pr(X = 0 \mid [Y_1 = y_1, Y_2 = y_2], Y_3 = y_3)$ is calculated according to [Hub02] as

$$p_{123} = (p_1 \otimes p_2) \otimes p_3. \tag{3.13}$$

Recursive application of the concept introduced in Equation (3.13) leads to

$$p_{12\ldots\delta_1} = (\ldots(p_1 \otimes p_2) \otimes \ldots p_{\delta_1}) = \frac{\prod_{i=1}^{\delta_1} p_i}{\prod_{i=1}^{\delta_1} p_i + \prod_{i=1}^{\delta_1} 1 - p_i}. \tag{3.14}$$

In Appendix A we show that the interval $[0, 1]$ forms an Abelian group under the normalized multiplication. As a consequence, the brackets in Equation (3.13) are not necessary.

Remark 3.1.1 *Another way of understanding the normalized multiplication is to consider it as an "optimal decoder for a repetition code". Multiple transmissions of a given symbol over δ_1 independent channels is equivalent to using a repetition code of rate $\frac{1}{\delta_1}$ and transmitting the symbols over a memoryless channel. Combining of probabilities considers all received symbols and exploits the coding rule completely by making use of the probability* $\Pr(X = 0 \mid Y_1 = y_1, Y_2 = y_2, \ldots, Y_{\delta_1} = y_{\delta_1})$.

Log-likelihood ratios (LLR)s [BGT93] are commonly used to implement the normalized multiplication (3.14). Let the operator $\ln(\cdot)$ denote the logarithm with base e. The LLR corresponding to probability p_i,

$$L_i = \ln\left(\frac{p_i}{1 - p_i}\right),$$

contains the same information as p_i does. It is easy to see that the log-likelihood ratio corresponding to $p_{12\ldots\delta_1}$, calculated in Equation (3.14), can be calculated by

$$L_{12\ldots\delta_1} = \sum_{i=1}^{\delta_1} L_i. \tag{3.15}$$

Proof 3.1.2 (Equation (3.15)) *Straightforward calculations show that Equation (3.15) corresponds to Equation (3.14).*

$$
\begin{aligned}
\ln\left(\frac{p_{12\ldots\delta_1}}{1-p_{12\ldots\delta_1}}\right) &= \sum_{i=1}^{\delta_1} \ln\left(\frac{p_i}{1-p_i}\right) \\
\frac{p_{12\ldots\delta_1}}{1-p_{12\ldots\delta_1}} &= \mathrm{e}^{\sum_{i=1}^{\delta_1} \ln\left(\frac{p_i}{1-p_i}\right)} \\
\frac{p_{12\ldots\delta_1}}{1-p_{12\ldots\delta_1}} &= \prod_{i=1}^{\delta_1}\left(\frac{p_i}{1-p_i}\right) \\
p_{12\ldots\delta_1} &= \frac{\prod_{i=1}^{\delta_1}\left(\frac{p_i}{1-p_i}\right)}{1+\prod_{i=1}^{\delta_1}\left(\frac{p_i}{1-p_i}\right)} \\
p_{12\ldots\delta_1} &= \frac{\prod_{i=1}^{\delta_1} p_i}{\prod_{i=1}^{\delta_1} p_i + \prod_{i=1}^{\delta_1}(1-p_i)}.
\end{aligned}
$$

■

Remark 3.1.3 *Although the log-likelihood representation is used frequently, it does not offer any advantages except for benefits with respect to the implementation. As shown above, the adding of log-likelihood ratios is equivalent to normalized multiplication of probabilities. This is stated in a mathematical way by the fact that there exists an isomorphism, i.e. a map which preserves sets and relations among elements [Wei02], between the groups* $([0,1],\otimes)$ *and* $(\mathbb{R},+)$.

Example 3.1.1 *(Transmission over the BEC) Probability combining is straightforward when the BEC is used as the channel model. Erased symbols correspond to the case that no information is received, thus* $\Pr(X=0 \mid Y_1 =?) = 0.5$. *If several channel outputs* $Y_1 = y_1, Y_2 = y_2, \ldots, Y_{\delta_1} = y_{\delta_1}$ *are observed and not all of them are erasures, combining of probabilities neglects the erasures and considers only one observation of multiply received symbols, i.e.*

$$
\Pr(X=0 \mid Y_1 =?, Y_2 =?, \ldots, Y_a = y_a, \ldots, Y_{\delta_1} = y_{\delta_1}) = \Pr(X=0 \mid Y_a = y_a), \quad y_a \in \{0,1\} \tag{3.16}
$$

The BEC model implies that $y_a = y_{\delta_1}$.

Example 3.1.2 *(Transmission over* δ_1 *real-valued AWGN channels) When transmitting over* δ_1 *real-valued AWGN channels, each with noise variance* σ_{R}^2, *combining of probabilities is conducted the following way. First, calculate* $\Pr(X=0 \mid Y_i = y_i)$, $i \in \{1,\ldots,\delta_1\}$, *given by*

$$
p_i = \frac{\mathrm{PDF}_{Y_i}(y_i \mid 0)\Pr(X=0)}{\mathrm{PDF}_{Y_i}(y_i \mid 0)\Pr(X=0) + \mathrm{PDF}_{Y_i}(y_i \mid 1)\Pr(X=1)}. \tag{3.17}
$$

For equiprobable input symbols and noise variance σ_{R}^2, one obtains

$$p_i = \frac{\frac{1}{\sqrt{2\pi\sigma_{\mathrm{R}}^2}}\mathrm{e}^{-\frac{(y_i-1)^2}{2\sigma_{\mathrm{R}}^2}}}{\frac{1}{\sqrt{2\pi\sigma_{\mathrm{R}}^2}}\mathrm{e}^{-\frac{(y_i-1)^2}{2\sigma_{\mathrm{R}}^2}} + \frac{1}{\sqrt{2\pi\sigma_{\mathrm{R}}^2}}\mathrm{e}^{-\frac{(y_i+1)^2}{2\sigma_{\mathrm{R}}^2}}} = \frac{1}{1+\mathrm{e}^{-\frac{2y_i}{\sigma_{\mathrm{R}}^2}}}. \tag{3.18}$$

Subsequently, a straightforward application of Equation (3.14) can be used to calculate $p_{12\ldots\delta_1}$.

Example 3.1.3 *(Example from different area of application) Calculating combined probabilities is of interest in many fields. In risk analysis, one aspires to calculate a probability which is based on the opinion of several experts. There exist several methods to do this, each with advantages when certain assumptions on the data are fulfilled. The aggregation of probabilities was already considered by Laplace while the calculation of a probability of the form $\Pr(X = 0 \mid Y_1 = y_1, Y_2 = y_2, \ldots, Y_{\delta_1} = y_{\delta_1})$ was studied in [Win68]. If the opinions of the experts are based on independent sources, normalized multiplication is used to calculate the combined opinion [CW90, CW99]. The fact that equal methods are used in a totally different field and that research has been conducted over such a long period of time emphasizes the importance of the discussed concept.*

3.2 Probabilities of Parity-Check Equations Being Satisfied

In this section we discuss δ_2 binary random sources X_1 to X_{δ_2}. The corresponding realizations are transmitted over a noisy channel. At the outputs of these channels, the realizations of the random variables Y_1 to Y_{δ_2} can be observed. Now we are interested in the probability that a parity-check equation is fulfilled. A parity-check equation is of the form

$$x_1 \oplus x_2 \oplus \ldots \oplus x_{\delta_2} = r, \tag{3.19}$$

where $x_1, x_2, \ldots, x_{\delta_2}$, and r are chosen from $\mathcal{X} \equiv \mathbb{F}_2$, $\mathbb{F}_q$ is the Galois field of order q, the symbol $\oplus$ denotes the addition in a Galois field (here: XOR operation), and where δ_2 is the number of symbols in the parity check. In order to use parity-check equations to describe linear codes [MS77, LC04], parity even is required and hence we set $r = 0$. This is a consequence of the fact that these codes are linear. The probability discussed here is the probability of a modulo-2 sum being zero when all channel outputs are known. It is given by

$$\Pr(x_1 \oplus x_2 \oplus \ldots \oplus x_{\delta_2} = 0 \mid Y_1 = y_1, Y_2 = y_2, \ldots Y_{\delta_2} = y_{\delta_2}). \tag{3.20}$$

Let us for a moment assume that $\delta_2 = 2$ is chosen. Then, $\Pr(X_1 \oplus X_2 = 0 \mid Y_1 = y_1, Y_2 = y_2)$ is considered. It is easy to see that $x_1 \oplus x_2 = 0$ if and only if (iff) $X_1 = 0$ and $X_2 = 0$ or $X_1 = 1$ and $X_2 = 1$. From this, it follows that the probability $\Pr(X_1 = 0, X_2 = 0 \cup X_1 = 1, X_2 = 1 \mid$

$Y_1 = y_1, Y_2 = y_2$) needs to be determined. Keep in mind that the events $(X_1 = 0, X_2 = 0)$ and $(X_1 = 1, X_2 = 1)$ are mutually exclusive. Furthermore assume that the events X_1 and X_2 are independent. Hence, one obtains

$$\begin{aligned}\Pr(X_1 \oplus X_2 = 0 \mid Y_1 = y_1, Y_2 = y_2) &= \Pr(X_1 = 0 \mid Y_1 = y_1)\Pr(X_2 = 0 \mid Y_2 = y_2) \\ &\quad + \Pr(X_1 = 1 \mid Y_1 = y_1)\Pr(X_2 = 1 \mid Y_2 = y_2).\end{aligned} \tag{3.21}$$

Using p_i, $i \in \{1, \ldots, \delta_2\}$ as defined in Equation (3.12), it is easy to see that

$$p_1 p_2 + (1 - p_1)(1 - p_2) = \frac{1}{2}(2p_1 - 1)(2p_2 - 1) + \frac{1}{2}. \tag{3.22}$$

This formula is referred to as the *Gallager formula* [Gal63, Ch. 4] and as "serial" information combining [LH06]. Let us now assume $\delta_2 > 2$. Then,

$$\Pr(X_1 \oplus \ldots \oplus X_{\delta_2} = 0 \mid Y_1 = y_1, Y_2 = y_2, \ldots, Y_{\delta_2} = y_{\delta_2}) = \frac{1}{2}\left(\prod_{i=1}^{\delta_2}(2p_i - 1)\right) + \frac{1}{2} \tag{3.23}$$

holds. This formula can be shown by induction [Wei02], where we closely follow [Hub02].

Proof 3.2.1 *Equation (3.23) obviously holds for $\delta_2 = 2$. Assuming that Equation (3.23) holds for $\delta_2 = \hat{\delta}_2 - 1$, it can be shown that it also holds for $\delta_2 = \hat{\delta}_2$:*

$$\begin{aligned}&\Pr(X_1 \oplus \ldots \oplus X_{\hat{\delta}_2} = 0 \mid Y_1 = y_1, Y_2 = y_2, \ldots, Y_{\hat{\delta}_2} = y_{\hat{\delta}_2}) = \\ &\tfrac{1}{2}\left(2\Pr(X_1 \oplus \ldots \oplus X_{\hat{\delta}_2 - 1} = 0 \mid Y_1 = y_1, \ldots, Y_{\hat{\delta}_2 - 1} = y_{\hat{\delta}_2 - 1}) - 1\right)\left(2p_{\hat{\delta}_2} - 1\right) + \tfrac{1}{2} = \\ &\tfrac{1}{2}\left(\prod_{i=1}^{\hat{\delta}_2 - 1}(2p_i - 1)\right)\left(2p_{\hat{\delta}_2} - 1\right) + \tfrac{1}{2} = \\ &\tfrac{1}{2}\left(\prod_{i=1}^{\hat{\delta}_2}(2p_i - 1)\right) + \tfrac{1}{2}.\end{aligned} \tag{3.24}$$

Thus, Equation (3.23) holds for all δ_2. ∎

Remark 3.2.2 *The Gallager formula can be used to decode an $[n, n-1, 2]$ single-parity-check (SPC) code in an optimal way. Assume that the RVs X_1, X_2, ..., X_{δ_2} are connected to each other by a parity-check equation, i.e.*

$$X_1 \oplus X_2 \oplus \ldots \oplus X_{\delta_2} = 0,$$

or, in other words, that the variable X_{δ_2} is given by

$$X_{\delta_2} = X_1 \oplus X_2 \oplus \ldots \oplus X_{\delta_2 - 1}. \tag{3.25}$$

The output of each random source X_i, $i = 1, \ldots, \delta_2$ cannot be observed directly, instead only a noisy representation Y_i, $i = 1, \ldots, \delta_2$ is available. This allows us to calculate

$$\Pr\left(X_{\delta_2} = 0 \mid [Y_j = y_j]_{j=1}^{\delta_2 - 1}\right),$$

i.e. a probability for a symbol which does not depend on the channel observation of the current symbol. This may be reasonable if such an observation is not available or an independent observation besides the direct one is required. We calculate

$$\Pr\left(X_{\delta_2} = 0 \mid [Y_j = y_j]_{j=1}^{\delta_2 - 1}\right) = \Pr\left(\bigoplus_{i=1}^{\delta_2 - 1} X_i = 0 \mid [Y_j = y_j]_{j=1}^{\delta_2 - 1}\right) = \frac{1}{2} \prod_{j=1}^{\delta_2 - 1} (2p_j - 1) + \frac{1}{2}. \tag{3.26}$$

Using the log-likelihood ratio $L_j = \ln\left(\frac{p_j}{1-p_j}\right)$ and the entity $2p_j - 1 = 2\frac{\mathrm{e}^{L_j}}{\mathrm{e}^{L_j}+1} - 1 = \tanh(L_j/2)$, Equation (3.26) can be rewritten as

$$2\Pr\left(X_{\delta_2} = 0 \mid [Y_j = y_j]_{j=1}^{\delta_2 - 1}\right) - 1 = \prod_{j=1}^{\delta_2 - 1} \tanh\left(\frac{L_j}{2}\right),$$

and consequently to

$$\tanh\left(L_{\delta_2}/2\right) = \prod_{j=1}^{\delta_2 - 1} \tanh\left(\frac{L_j}{2}\right). \tag{3.27}$$

In contrast to the normalized multiplication, cf. Equation (3.14), log-likelihood ratios are not a convenient representation for describing the calculation in Equation (3.26). In this thesis, we will consistently use probabilities. This keeps the presentation for both types of operations simple.

Example 3.2.1 *(Transmission over the BEC) For the BEC, the task of a parity check is limited to recovering at most one erasure. This can be accomplished if there exists exactly one erased symbol in the considered parity check. We deduct this from Equation (3.26). Let δ_2 be the erased position which is to be recovered. If another position is erased as well, i.e. the corresponding probability equals $1/2$, the right-hand side of Equation (3.26) equals $1/2$.*

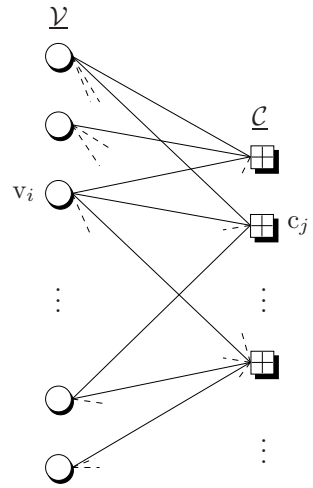
Chapter 4

Belief-Propagation Decoding

Channel coding schemes add redundancy to the transmitted data and use the knowledge on the redundancy at the receiver in order to prevent errors or recover erasures. For this task, a *decoding algorithm* is required at the receiver. We discuss the belief-propagation (BP) algorithm. This decoding algorithm is widely used for iterative decoding of LDPC codes, but can also be applied to any other block code. Belief propagation is a special type of *message-passing* algorithm [Mac03, p. 241]. We cover the BP algorithm from two different points of view. First, from the channel decoding point of view, where we consider decoding of an LDPC code as iterative decoding of basic constituent codes in an optimal way. Information exchange allows to solve the global decoding problem in this setting. Second, we consider this algorithm as a solution to a marginalization problem which makes use of distributed hardware and subsequently specialize this solution to the channel coding case. These two perceptions usually lead to quite different forms of presentation in the literature [Hub02, Mac03]. Besides providing a rigorous introduction of the algorithm, it is our intention to show the equivalence of these two perceptions.

4.1 Bipartite Graphs

Belief-propagation decoders work on an $m \times n$ parity-check matrix $\boldsymbol{H}$ of a given linear block code. The way in which this matrix is chosen for a given channel code is detailed in Chapter 5. At this moment, we are content with the fact that $\boldsymbol{H}$ is one representation of a code. This matrix corresponds unambiguously to a bipartite *Tanner graph* [Tan81], henceforth denoted by $\mathcal{G}$. This graph consists of *variable nodes* v_i, $i = 1, \ldots, n$, *check nodes* c_j, $j = 1, \ldots, m$, and *edges* connecting two nodes of different type. The set of variable nodes will henceforth be denoted by $\underline{\mathcal{V}}$, while $\underline{\mathcal{C}}$ will refer to the set of check nodes. The bar in the notation is used to remind the reader that the variables are defining sets of a graph. Note that each check node in combination with the edges connected to it corresponds to a row of $\boldsymbol{H}$ and that each variable node along with the edges connected to it represents a column of $\boldsymbol{H}$. An edge connecting variable node v_i and check node c_j corresponds to $H(i, j) = 1$. Figure 4.1 shows a Tanner graph together with its corresponding parity-check matrix representation.



$$\boldsymbol{H} = \begin{array}{c} \text{indices} \\ j \\ \downarrow \end{array} \overset{\begin{array}{cccccc} & & i & \rightarrow & & \end{array}}{\begin{bmatrix} 1 & 1 & 1 & \dots & & \\ 1 & 0 & 1 & \dots & 1 & 0 \\ & \vdots & & & & \vdots \\ & 1 & & & 1 & 1 \\ & \vdots & & & & \vdots \end{bmatrix}}$$

Figure 4.1: Bipartite Tanner graph and corresponding parity-check matrix.

4.2 Belief-Propagation from Channel Decoding Point of View

Assume that a vector of encoded symbols $\boldsymbol{x}$ is signaled over a noisy channel and let us restrict our attention to a single encoded symbol x_i, $i = 1, \dots, n$. The corresponding variable node in the Tanner graph is v_i. This node is an abstract object to visualize the position of x_i in $\boldsymbol{x}$.

With complete information on the channel code readily available, the received vector $\boldsymbol{y}$ allows for multiple independent conclusions on x_i. This is detailed in the following. First, a direct observation from the channel is available at the decoder by considering only y_i. If we assume the channel to be memoryless, y_i depends on the encoded symbol x_i and the corresponding distortion. We refer to the corresponding probabilities as *intrinsic* probabilities [LH06] and denote them by $p_i^{\text{int}} = \Pr(X_i = 0 \mid Y_i = y_i)$.

The code rules allow to deduct multiple indirect observations from the remaining received symbols. We refer to probabilities for the event $X_i = 0$ which depend only on indirect observations (y_j, $j = 1, \dots, n$, $j \neq i$) as *extrinsic* probabilities. These probabilities are denoted by p^{ext} if information is being sent from the variable nodes and q^{ext} if the information is being sent out by the check nodes. The presence of direct and indirect observations of a given event resembles the scenario of a repetition code. In a more general way it can also be understood as multiple transmissions of data over independent channels, cf. Chapter 2. All incoming messages at variable node v_i carry information on the encoded symbol x_i, and were distorted independently. The optimal way to decode a repetition code is by means of the normalized multiplication as discussed in Section 3.1. The belief-propagation algorithm applies this operation at each variable node.

Let us now discuss one check node and the edges connected to it. This node represents one of m constraints that have to be fulfilled by all valid codewords. This single parity-check equation can be regarded as a super-code of the deployed LDPC code. A single parity-check can also be

seen as a parity-check matrix of dimension $1 \times n$, and this matrix corresponds to an SPC code. The optimal way to decode an SPC code is to retrieve soft information on whether the check is satisfied or not. This task is accomplished in an optimal way via the Gallager-formula, cf. Section 3.2.

The decoding procedure of an LDPC code can be divided into very basic local operations, which can be carried out in an optimal way. This includes the normalized multiplication at the variable nodes and the application of the Gallager formula at the check nodes. The idea of the belief-propagation algorithm is to apply these procedures in a repeated manner. Using the local functions for one type of node and sending updated messages over all edges to the nodes of different type is henceforth referred to as one *half-iteration*. We refer to the concatenation of two subsequent half-iterations as one iteration. Here, we assume that messages are first sent from the variable nodes to the check nodes and then vice versa. This iterative concept allows for data exchange between the local functions. As all subtasks are solved in an optimal manner, an optimal solution to the global problem may seem expectable. However, the optimality of the local operations is only given if the graph is a tree, i.e. has no closed loops (cycles). This is not the case for any practical LDPC code. However, we will observe that decoding with desirable performance results is still possible. One very important element for this success is the *extrinsic principle* which mitigates the feedback effect.

4.2.1 Extrinsic Principle

In order to demonstrate the effect of the extrinsic principle, we concentrate on one variable node v_i and one check node c_j, connected to each other by an edge of the Tanner graph. The neighborhood of c_j, i.e. the set of indices of all variable nodes that c_j is connected to, is denoted by $\mathcal{N}(\mathrm{c}_j)$, where we assume that $i \in \mathcal{N}(\mathrm{c}_j)$.

Let us briefly discuss the proceeding when information from the check nodes is sent to the variable nodes. Variable node v_i, $i = 1, \ldots, n$, considers the incoming information as extrinsic information. The node combines all information on x_i from the connected check nodes and the intrinsic information received from the channel into one probability. If the extrinsic principle is obeyed, we refer to this combined probability as the a-posteriori probability (APP). In that case, the APP is specified as

$$p_i^{\mathrm{app}} = p_i^{\mathrm{int}} \otimes \boxtimes_{i' \in \mathcal{N}(\mathrm{v}_i)} q_{ii'}^{\mathrm{ext}}. \tag{4.1}$$

The double-index notation of $q_{ii'}^{\mathrm{ext}}$ expresses that the extrinsic information is sent from check node c_i to variable node $\mathrm{v}_{i'}$. A-posteriori probabilities are denoted by p^{app} when information is sent from a variable node and q^{app} if it originates from a check node.

In the following, we discuss a counter-example, i.e. assume that the check node does not obey the extrinsic principle and study the effect of this assumption. Figure 4.2 shows the variable node sending its *intrinsic* information in the very first half-iteration to the check node. The

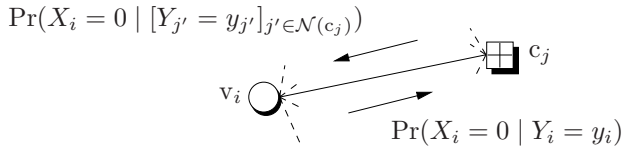


Figure 4.2: Check node and variable node, check node is not following extrinsic principle.

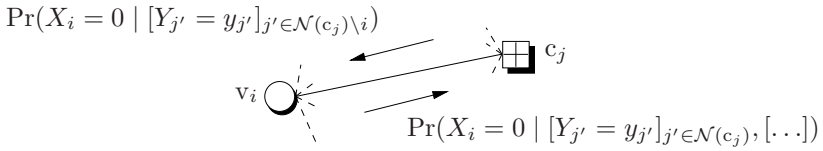


Figure 4.3: Check node and variable node, variable node is not following extrinsic principle.

latter one does not obey the extrinsic principle and sends back information that depends on all received values within the neighborhood, including y_i. As the assumption that $q_{ii'}^{\mathrm{ext}}$ is indeed an extrinsic probability does not hold, the evaluation of Equation (4.1) considers the observation $Y_i = y_i$ twice: once in p_i^{int} and a second time in one of the incoming probabilities $q_{ii'}^{\mathrm{ext}}$. The independency assumption which implies that the normalized multiplication is optimal for combining probabilities is not fulfilled in this case.

An equivalent situation arises when the variable node does not follow the extrinsic principle. In this case, the variable node sends information which it obtained from $\mathcal{N}(\mathrm{c}_j)$ together with further information, denoted by $\Pr(X_i = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)}, [\ldots])$ to c_j. Figure 4.3 presents this situation. The check node cannot calculate the probability $\Pr\left(X_i = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)}\right)$ by application of the Gallager formula as the independency assumption (cf. Section 3.2) of the incoming data is not fulfilled.

Let us now show how the extrinsic principle mitigates this effect.

- **Extrinsic principle at the check nodes**. We assume that the check nodes have collected channel information from all variable nodes in the neighborhood within the first iteration. This means that variable node v_i has sent $\Pr(X_i = 0 \mid Y_i = y_i)$ to all its neighboring check nodes. With this information, check node c_j can calculate the probability for the check node being satisfied using the incoming information from all edges but one, cf. Equation (3.26). Let this edge connect check node c_j to variable node v_i. The probability calculated by the check node reads

$$\Pr\left(X_i = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j) \setminus i}\right),$$

  cf. Equation (3.25) and Equation (3.26). Hence, it provides information on X_i which is independent from the information currently available at v_i. The check node calculates this probability using

$$\Pr\left(X_i = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)\setminus i}\right) = \Pr\left(\bigoplus_{j' \in \mathcal{N}(\mathrm{c}_j)\setminus i} X_{j'} = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)\setminus i}\right).$$

Assuming that the realization of X_i is given, the message from the check node can be understood as an independent observation at v_i. Independent observations of that type are required for the normalized multiplication, cf. Equation (3.4). The variable node considers all these observations in order to update its a-posteriori probability and the corresponding extrinsic probabilities.

- **Extrinsic principle at the variable nodes**. We assume that all variable nodes have received information from the check nodes prior to this step. The extrinsic principle instructs variable node v_i to send all its information to check node c_j, except the information v_i has obtained from c_j. To this end, variable node v_i performs combining of probabilities of all incoming messages, including the message from c_j, i.e. $\Pr\left(X_i = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)\setminus i}\right)$. The result is then combined with the information obtained from the channel observation, $\Pr(X_i = 0 \mid Y_i = y_i)$. Before sending information to c_j, variable node v_i "divides" the information obtained from c_j from the remaining information by performing a normalized multiplication with the complementary probability $1 - \Pr\left(X_i = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)\setminus i}\right)$. This operation is denoted by the operator $\oslash$, where

$$a \oslash b = \frac{a/b}{a/b + (1-a)/(1-b)} = \frac{a \cdot (1-b)}{a \cdot (1-b) + (1-a) \cdot b},$$

cf. [Hub02].

Following the extrinsic principle both at the variable nodes and the check nodes prevents information from returning to its originator within one iteration.

4.2.2 Limits of the Extrinsic Principle

Implementations of the BP algorithm used for decoding purposes follow the extrinsic principle. Nevertheless, there exist feedback paths in the Tanner graph that are not considered by the extrinsic principle. We follow the path of the observation $Y_1 = y_1$, indicated by the arrows in Figure 4.4.

Example 4.2.1 *Given is the extract of a Tanner graph shown in Figure 4.4. Let us analyze how information depending on the received value y_1 returns to v_1 within four subsequent half-iterations of the decoder.*

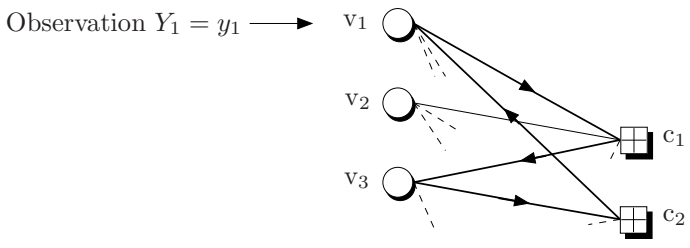


Figure 4.4: Detail of a bipartite graph showing a closed loop of length 4, visualized by the bold line.

- ***Half-iteration*** 1***:*** *Variable node* v_1 *sends its intrinsic probability* $\Pr(X_1 = 0 \mid Y_1)$ *to check node* c_1 *(and other check nodes).*

- ***Half-iteration*** 2***:*** *Due to the extrinsic principle,* c_1 *sends a probability that depends on* y_1 *only to* v_2 *and* v_3*. We focus on the probability that is sent to* v_3*. This probability reads* $\Pr\left(X_3 = 0 \mid [Y_j = y_j]_{j \in \mathcal{N}(\mathrm{c}_1) \backslash 3}\right)$ *and is based on the observation* y_1*.*

- ***Half-iteration*** 3***:*** *Variable node* v_3 *combines the information it obtained from* c_1 *with information from all other check nodes. The extrinsic principle instructs* v_3 *to send information retrieved from* c_1 *only to* c_2*. The corresponding message sent to* c_2 *is* $\Pr(X_3 = 0 \mid [Y_j = y_j]_{j \in \mathcal{N}(\mathrm{c}_1) \backslash 3}, [\ldots])$*.*

- ***Half-iteration*** 4***:*** *It is now the task of check node* c_2 *to send the information obtained from* v_3 *and others except* v_1 *to* v_1*. Although eliminating the information directly obtained from* v_1*, the extrinsic principle cannot help that information coming from* v_3*,* $\Pr(X_3 = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_1) \backslash 3}, [\ldots])$ *includes knowledge on the event* $Y_1 = y_1$*. Check node* c_2 *sends* $\Pr\left(\sum_{j' \in \mathcal{N}(\mathrm{c}_2) \backslash 1} X_1 = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_2) \backslash 1}\right)$ *to* v_1*. At this point, variable node* v_1 *obtains a probability depending on* y_1 *over the edges of the Tanner graph. We state that the calculation of the a-posteriori probability and all subsequent operations cannot be conducted in an optimal way any more.*

These observations lead to the following definitions.

Definition 4.2.1 *The length of the shortest cycle in a given graph* $\mathcal{G}$*, denoted by* $\mathrm{gi}(\mathcal{G})$ *is called the girth of* $\mathcal{G}$*.*

Definition 4.2.2 *The local girth* $\mathrm{lgi}(\mathcal{G}, \mathrm{v}_i)$ *of a graph* $\mathcal{G}$ *and a variable node* v_i *is the shortest cycle within the graph that passes through* v_i*.*

Example 4.2.2 *(Continuation of Example 4.2.1) An upper bound on the girth of the graph shown in Figure 4.4 is four. Due to the extrinsic principle, the smallest possible girth in any Tanner graph reads four, and hence the girth is exactly four. Tanner graphs should be optimized in such a way that the girth is maximized. This is accomplished by avoiding small cycles.*

It is common belief that BP decoding implements an optimal decoding scheme if no more than $\mathrm{gi}(\mathcal{G})/2$ iterations are performed. This is only true for a variable node if all connected parity checks do not contain any further variable nodes in common[1]. This is a condition that is usually not fulfilled. Figure 4.4 shows this exemplary. Already in the second half-iteration, v_3 combines information from c_1 and c_2, both depending on the observation of the event $Y_1 = y_1$. Even when deploying the extrinsic principle, the BP algorithm is suboptimal. However, its application is of interest, as we are only interested in a binary quantization of the resulting probabilities.

The BP algorithm based on the normalized multiplication at the variable nodes and the Gallager formula at the check nodes is given in Algorithm 4.1. It is the purpose of the algorithm to recover the vector $\boldsymbol{x}$ of length n (encoded symbols). The algorithm follows the *flooding schedule* [MB01b] which essentially states that all variable nodes and all check nodes operate in alternating turns. In other words, at a given time step information is sent over the edges only in one direction (left-to-right or right-to-left).

Let us now present pseudocode for the belief-propagation algorithm. For a rigorous exposition of the algorithm we introduce the operation of transposing a matrix, denoted by $(\cdot)^{\mathsf{T}}$. Henceforth, we denote the current iteration by I, and the maximum number of iterations by $I_{\max}$. The double index of the variables $p^{\mathrm{ext}}_{j'j}$ states that the information $p_{j'j}$ is sent from variable node $\mathrm{v}_{j'}$ to check node c_j. The symbol $\odot$ denotes the multiplication in a Galois field, here $\mathbb{F}_2$, while the function bi2re allows us to transition from $\mathbb{F}_2$ to $\mathbb{R}$. We define it as $\mathrm{bi2re}(\cdot) : \mathbb{F}_2 \to \mathbb{R}$, a function which maps $0 \in \mathbb{F}_2 \to 0 \in \mathbb{R}$ and $1 \in \mathbb{F}_2 \to 1 \in \mathbb{R}$. Algorithm 4.1 outlines the steps of the BP algorithm.

4.3 Belief-Propagation from Algorithmic Point of View

Belief-propagation decoding is often referred to as sum-product decoding [CRU01] and message-passing decoding [RU01]. It is our intention to clarify the heritage of these denominations and classify the relations between these algorithms. The sum-product algorithm is a general algorithm for efficiently computing marginalizations of factorizable functions. This algorithm is intended to work on tree-like factor graphs. In channel coding however, it is commonly used on graphs which do not fulfill this property ("loopy decoding", [Mac03]). The sum-product algorithm is a special message-passing algorithm. This class of algorithms is identified by the property that small distributed processors conduct easy tasks. Messages are passed between these local processes and with repeated application of this scheme, a more complex and global problem is solved. The min-sum algorithm [Mac03] is another representative of message-passing algorithms.

In the following, we observe that the application of the sum-product algorithm to the channel decoding problem is equivalent [Mac03, p. 330] to the belief-propagation approach discussed in Section 4.2.

[1]Note that this property is included in the tree assumption which implies optimal decoding results by the BP decoder. A tree is a simple graph [Wei02] and hence all connected nodes are connected by at most one edge.

Algorithm 4.1 Belief-Propagation using normalized multiplication and the Gallager formula.

Input: $\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, optionally p^{ext}
Output: $\hat{\boldsymbol{x}}$, success_flag, p^{ext}
Function call: $[\hat{\boldsymbol{x}}, \text{success\_flag}, p^{\text{ext}}]$=bp_decode($\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, $\langle p^{\text{ext}} \rangle$)

1: **/* Initialize variable node probabilities */**
2: $p_i^{\text{int}} = \Pr(X_i = 0 \mid Y_i = y_i), i = 1, \ldots, n$
3: $p_i^{\text{app}} = p_i^{\text{int}}, i = 1, \ldots, n, \boldsymbol{p}^{\text{int/app}} = [p_1^{\text{int/app}}, \ldots, p_n^{\text{int/app}}]$
4: **if** p^{ext} **not defined then**
5: $\quad p_{ij}^{\text{ext}} = p_i^{\text{int}}, i = 1, \ldots, n, j \in \mathcal{N}(\mathrm{v}_i)$
6: **end if**
7: **while** $I < I_{\max}$ **and** $\lceil \boldsymbol{p}^{\text{app}} \rfloor \odot \boldsymbol{H}^{\mathsf{T}} \neq \boldsymbol{0}$ **do**
8: $\quad$ **/* Process information at the check nodes */**
9: $\quad$ **for** $j = 1, \ldots, m$ **do**
10: $\quad\quad$ **/* Calculate a-posteriori information */**
11: $\quad\quad q_j^{\text{app}} = \Pr\left(\boxplus_{j' \in \mathcal{N}(\mathrm{c}_j)} X_{j'} = 0 \mid [Y_{j'} = y_{j'}]_{j' \in \mathcal{N}(\mathrm{c}_j)} \right) = \frac{1}{2} \prod_{j' \in \mathcal{N}(\mathrm{c}_j)} \left(2 p_{j'j}^{\text{ext}} - 1\right) + \frac{1}{2}$
12: $\quad\quad$ **/* Calculate extrinsic information for each connected variable node */**
13: $\quad\quad$ **for** $j' \in \mathcal{N}(\mathrm{c}_j)$ **do**
14: $\quad\quad\quad q_{jj'}^{\text{ext}} = q_j^{\text{app}} / (2 p_{j'j}^{\text{ext}} - 1)$
15: $\quad\quad$ **end for**
16: $\quad$ **end for**
17: $\quad$ **/* Process information at the variable nodes */**
18: $\quad$ **for** $i = 1, \ldots, n$ **do**
19: $\quad\quad$ **/* Calculate a-posteriori information */**
20: $\quad\quad p_i^{\text{app}} = \Pr(X_i = 0 \mid \ldots) = p_i^{\text{int}} \otimes \boxtimes_{i' \in \mathcal{N}(\mathrm{v}_i)} q_{i'i}^{\text{ext}}$
21: $\quad\quad$ **/* Calculate extrinsic information for each connected check node */**
22: $\quad\quad$ **for** $i' \in \mathcal{N}(\mathrm{v}_i)$ **do**
23: $\quad\quad\quad p_{ii'}^{\text{ext}} = p_i^{\text{app}} \otimes (1 - q_{i'i}^{\text{ext}})$
24: $\quad\quad$ **end for**
25: $\quad$ **end for**
26: $\quad I = I + 1$
27: **end while**
28: **/* Hard decision using a-posteriori info */**
29: $\hat{\boldsymbol{x}} = \lceil \boldsymbol{p}^{\text{app}} \rfloor$
30: success_flag $= \sum \mathrm{bi2re}(\hat{\boldsymbol{x}} \odot \boldsymbol{H}^{\mathsf{T}})$

Let us now review the sum-product algorithm in more detail and establish its connection to channel decoding. We consider the problem of calculating the value of a function which can be factorized [KFL01, Mac03]. We consider a general function $\mathrm{f}(\boldsymbol{x})$ with $\boldsymbol{x} = [x_1, \ldots, x_n]$. Assume that the function can be factorized according to

$$\mathrm{f}(\boldsymbol{x}) = \prod_{j=1}^{m} \mathrm{f}_j(\boldsymbol{x}(\mathcal{P}_j)), \tag{4.2}$$

where $\mathcal{P} := \{1, \ldots, n\}$ holds and $\mathcal{P}_j$ is a subset of $\mathcal{P}$. The notation $\boldsymbol{x}(\mathcal{P}_j)$ states that we consider only the elements of $\boldsymbol{x}$ which are indexed by $\mathcal{P}_j$. The variable m was originally introduced as

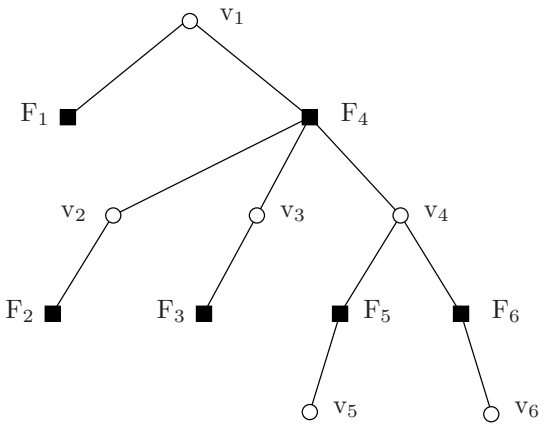


Figure 4.5: Factor graph for Equation (4.4).

the number of checks in a Tanner graph. We use it in this context to denote the number of local functions. This is due to the fact that check nodes and local functions correspond to each other in this investigation. In the following, we define a factor graph for a given function.

Definition 4.3.1 *([KFL01]) A factor graph is a bipartite graph which expresses the structure of the factorization of a function as given in Equation (4.2). A factor graph has a variable node* v_i *for each variable* x_i*,* $i \in \{1, \ldots, n\}$*, a factor node* F_j *for each local function* f_j*,* $j \in \{1, \ldots, m\}$*, and an edge connecting variable node* v_i *to factor node* F_j *iff* x_i *is an argument of* f_j*.*

Example 4.3.1 *Assume that the function* $\mathrm{f}(\boldsymbol{x})$ *is factorizable in the following way:*

$$\mathrm{f}(\boldsymbol{x}) = \mathrm{f}_1(x_1) \cdot \mathrm{f}_2(x_2) \cdot \mathrm{f}_3(x_3) \cdot \mathrm{f}_4(x_1, x_2, x_3, x_4) \cdot \mathrm{f}_5(x_4, x_5) \cdot \mathrm{f}_6(x_4, x_6)$$

We are interested in calculating a marginal function, e.g. $\sum_{\sim x_i} \mathrm{f}(\boldsymbol{x})$*, where the notation "*$\sim x_i$*" denotes that the summation is performed over all variables except* x_i*. Further it is assumed that* x_i *is an element of the vector* $\boldsymbol{x}$*. We obtain*

$$\sum_{\sim x_1} \mathrm{f}(\boldsymbol{x}) = \sum_{\sim x_1} \mathrm{f}_1(x_1) \cdot \mathrm{f}_2(x_2) \cdot \mathrm{f}_3(x_3) \cdot \mathrm{f}_4(x_1, x_2, x_3, x_4) \cdot \mathrm{f}_5(x_4, x_5) \cdot \mathrm{f}_6(x_4, x_6) \tag{4.3}$$

We introduce the notation $\mathrm{f}^{[\mathrm{M}]}(x_i) = \sum_{\sim x_i} \mathrm{f}(\boldsymbol{x})$ *in order to emphasize that the marginalized function differs from the original one. For an efficient calculation, we rewrite Equation (4.3) as*

$$\mathrm{f}^{[\mathrm{M}]}(x_1) = \mathrm{f}_1(x_1) \cdot \sum_{\sim x_1} \left(\mathrm{f}_4(x_1, x_2, x_3, x_4) \cdot \mathrm{f}_2(x_2) \cdot \mathrm{f}_3(x_3) \cdot \sum_{\sim x_4} \mathrm{f}_5(x_5, x_4) \cdot \sum_{\sim x_4} \mathrm{f}_6(x_4, x_6) \right). \tag{4.4}$$

Figure 4.5 presents the factor graph of the function $\mathrm{f}(\boldsymbol{x})$*, where the graph was drawn in such a way that* v_1 *marks the root. It can be used to realize the calculation of Equation (4.4) by*

following the rules of the sum-product algorithm for the calculation of the marginalization for one variable, to be introduced shortly. In practice, the sum-product algorithm does not calculate all marginalizations separately but in a more sophisticated way, discussed in the following.

In order to calculate the marginalization of the function, the following update rules have to be realized. Each variable node sends the product of all incoming messages from the child nodes, which lie beneath the node, to the parent of this node, which lies atop. Each factor node multiplies all incoming messages with the function value. The outgoing message depends on the receiving node. If the message is sent to parent node v_i, a summation over all variables except for v_i is performed.

Let us now discuss how the factor graph can be used to calculate any marginalization of $\mathrm{f}(\boldsymbol{x})$, i.e. $\sum_{\sim x_i} \mathrm{f}(\boldsymbol{x})$, $i = 1, \ldots, n$. It would be a straightforward approach to calculate all these marginalizations in a serial manner, thus rewriting Equation (4.4) such that x_1 to x_n appear as the argument on the left-hand side. However, there exists a more elegant way of solving this problem which is formed by overlaying all these graphs, resulting in one non-directed graph. For this method, there is no distinction into child nodes and parent nodes, but each node plays both roles in turn during the decoding process. Whenever a node has obtained information from all edges except one, it passes corresponding information to the node from which no information was accepted so far. Subsequently, it waits for a message from this particular node and updates all remaining nodes connected to it.

Definition 4.3.2 *([KFL01]) Let $\mu_{\mathrm{v}_i \to \mathrm{F}_j}$ be the message sent from variable node v_i to factor node F_j and $\mu_{\mathrm{F}_j \to \mathrm{v}_i}$ be the message sent from factor node F_j to variable node v_i. The messages $\mu_{\mathrm{v}_i \to \mathrm{F}_j}$ are initialized by* 1*. The updating procedures read*

$$\mu_{\mathrm{v}_i \to \mathrm{F}_j} = \prod_{j' \in \mathcal{N}(\mathrm{v}_i) \setminus j} \mu_{\mathrm{F}_{j'} \to \mathrm{v}_i}, \tag{4.5}$$

and

$$\mu_{\mathrm{F}_j \to \mathrm{v}_i} = \sum_{\sim x_i} \left(\mathrm{f}_j(\boldsymbol{x}(\mathcal{P}_j)) \prod_{i' \in \mathcal{N}(\mathrm{F}_j) \setminus i} \mu_{\mathrm{v}_{i'} \to \mathrm{F}_j} \right). \tag{4.6}$$

This algorithm starts at the leaves of the tree and proceeds until each edge has transported information in each direction. Then, all variable and function nodes have compiled all required information and further steps are not required. As only summation and multiplication is required, this algorithm is referred to as the **sum-product** algorithm.

Let us now return to the example used to explain the sum-product algorithm.

Example 4.3.2 *(Continuation of Example 4.3.1) Consider the graphical representation of Equation (4.4) given in Figure 4.5. The straightforward approach to calculate all marginal distributions*

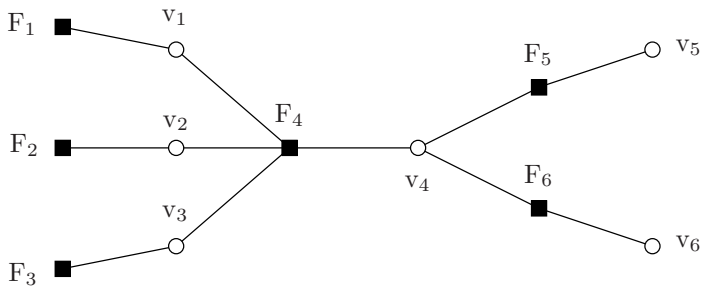


Figure 4.6: Factor graph from Figure 4.5, no specific root exposed.

would involve redrawing Figure 4.5 with the root node varying from v_1 *to* v_6*. Figure 4.6 allows to obtain these distributions with one graph and fewer calculations. The required operations are listed in the following.*

1. *Send* $\mu_{\mathrm{F}_1\to\mathrm{v}_1} = \mathrm{f}_1(x_1)$, $\mu_{\mathrm{F}_2\to\mathrm{v}_2} = \mathrm{f}_2(x_2)$, $\mu_{\mathrm{F}_3\to\mathrm{v}_3} = \mathrm{f}_3(x_3)$, *as well as* $\mu_{\mathrm{v}_5\to\mathrm{F}_5} = 1$, $\mu_{\mathrm{v}_6\to\mathrm{F}_6} = 1$.

2. *Send* $\mu_{\mathrm{v}_1\to\mathrm{F}_4} = \mathrm{f}_1(x_1)$, $\mu_{\mathrm{v}_2\to\mathrm{F}_4} = \mathrm{f}_2(x_2)$, $\mu_{\mathrm{v}_3\to\mathrm{F}_4} = \mathrm{f}_3(x_3)$, *as well as* $\mu_{\mathrm{F}_5\to\mathrm{v}_4} = \sum_{\sim x_4} \mathrm{f}_5(x_4, x_5)$, $\mu_{\mathrm{F}_6\to\mathrm{v}_4} = \sum_{\sim x_4} \mathrm{f}_6(x_4, x_6)$.

3. *Send* $\mu_{\mathrm{F}_4\to\mathrm{v}_4} = \sum_{\sim x_4} \mathrm{f}_1(x_1)\mathrm{f}_2(x_2)\mathrm{f}_3(x_3)\mathrm{f}_4(x_1, x_2, x_3, x_4)$, $\mu_{\mathrm{v}_4\to\mathrm{F}_4} = \mu_{\mathrm{F}_5\to\mathrm{v}_4}\mu_{\mathrm{F}_6\to\mathrm{v}_4}$.

4. *Send* $\mu_{\mathrm{F}_4\to\mathrm{v}_1} = \sum_{\sim x_1} \mathrm{f}_4(x_1, x_2, x_3, x_4)\mathrm{f}_2(x_2)\mathrm{f}_3(x_3)\mu_{\mathrm{v}_4\to\mathrm{F}_4}$,

   $\mu_{\mathrm{F}_4\to\mathrm{v}_2} = \sum_{\sim x_2} \mathrm{f}_4(x_1, x_2, x_3, x_4)\mathrm{f}_1(x_1)\mathrm{f}_3(x_3)\mu_{\mathrm{v}_4\to\mathrm{F}_4}$,

   $\mu_{\mathrm{F}_4\to\mathrm{v}_3} = \sum_{\sim x_3} \mathrm{f}_4(x_1, x_2, x_3, x_4)\mathrm{f}_1(x_1)\mathrm{f}_2(x_2)\mu_{\mathrm{v}_4\to\mathrm{F}_4}$,

   as well as

   $\mu_{\mathrm{v}_4\to\mathrm{F}_5} = \mu_{\mathrm{F}_4\to\mathrm{v}_4}\mu_{\mathrm{F}_6\to\mathrm{v}_4}$,

   $\mu_{\mathrm{v}_4\to\mathrm{F}_6} = \mu_{\mathrm{F}_4\to\mathrm{v}_4}\mu_{\mathrm{F}_5\to\mathrm{v}_4}$

5. *Send* $\mu_{\mathrm{v}_1\to\mathrm{F}_1} = \mu_{\mathrm{F}_4\to\mathrm{v}_1}$,

   $\mu_{\mathrm{v}_2\to\mathrm{F}_2} = \mu_{\mathrm{F}_4\to\mathrm{v}_2}$,

   $\mu_{\mathrm{v}_3\to\mathrm{F}_3} = \mu_{\mathrm{F}_4\to\mathrm{v}_3}$,

   as well as

   $\mu_{\mathrm{F}_5\to\mathrm{v}_5} = \sum_{\sim x_5} \mathrm{f}_5(x_4, x_5)\mu_{\mathrm{v}_4\to\mathrm{F}_5}$,

   $\mu_{\mathrm{F}_6\to\mathrm{v}_6} = \sum_{\sim x_6} \mathrm{f}_6(x_4, x_6)\mu_{\mathrm{v}_4\to\mathrm{F}_6}$

After these five steps, the algorithm terminates. It is now straightforward to see that the marginal distributions that depend only on x_i*,* $i \in 1, \ldots, 6$*, can be obtained by multiplication of all messages sent to* v_i*. Let us show this exemplary for the variable node* v_5*. This result reads*

$$\mathrm{f}^{[\mathrm{M}]}(x_5) = \mu_{\mathrm{F}_5 \to \mathrm{v}_5} = \sum_{\sim x_5} \mathrm{f}_5(x_4, x_5) \cdot \sum_{\sim x_4} \mathrm{f}_1(x_1) \cdot \mathrm{f}_2(x_2) \cdot \mathrm{f}_3(x_3) \cdot \mathrm{f}_4(x_1, x_2, x_3, x_4) \cdot \sum_{\sim x_4} \mathrm{f}_6(x_4, x_6).$$

In order to validate this result, we recalculate it by performing the direct summation operations over $\mathrm{f}(\boldsymbol{x})$*. We obtain*

$$\mathrm{f}^{[\mathrm{M}]}(x_5) = \sum_{x_4} \mathrm{f}_5(x_4, x_5) \cdot \sum_{x_1} \sum_{x_2} \sum_{x_3} \mathrm{f}_1(x_1) \cdot \mathrm{f}_2(x_2) \cdot \mathrm{f}_3(x_3) \cdot \mathrm{f}_4(x_1, x_2, x_3, x_4) \cdot \sum_{x_6} \mathrm{f}_6(x_4, x_6).$$

4.3.1 Decoding LDPC Codes with the Sum-Product Algorithm

Let us now use the sum-product algorithm for the more challenging task of decoding LDPC codes. Here, we are interested in calculating marginal distributions $\Pr(X_i = 0 \mid [Y_1, \ldots, Y_n] = [y_1, \ldots, y_n])$, $i = 1, \ldots, n$, which inform us how likely it is that a transmitted codesymbol x_i is zero when the received vector of length n and the code constraints are known. If we were able to calculate the probability $\Pr(X_i = 0 \mid [Y_1, \ldots, Y_n] = [y_1, \ldots, y_n])$ for all $i = 1, \ldots, n$ exactly, we could decide for the most probable codeword by choosing the most probable encoded symbol for each value of i. This cannot be done in an optimal way with the approach introduced here due to the closed loops in the Tanner graph. However, remember that we focus only on a binary decision of x_i as a result. Obtaining this result correctly is still possible if the deviation of the resulting probabilities is not too high. This motivates the use of the algorithm despite of the fact that it is not optimal. When using the sum-product algorithm introduced in Section 4.3 for the purpose of decoding LDPC codes, the following adaptations need to be done.

- The messages μ are identified as the probabilities of the bits corresponding to the given variable nodes. Modifications using log-likelihood ratios exist.
- As the graph contains cycles, the initialization at the leaves described above is not possible. Instead, it is assumed at the beginning of the process that all variable nodes have heard from all incoming edges and are hence ready to communicate to any check node.
- The function f determines whether a given codeword belongs to the code, i.e. the function f checks whether all parity-check equations are fulfilled. This function can be factorized into a product of the functions which resemble the single parity checks defining this code. In factorized form, f can be given as

$$\mathrm{f}(\boldsymbol{x}) = \prod_{j=1}^{m} \mathrm{bi2re}\left(1 - \bigoplus_{i \in \mathcal{N}(\mathrm{F}_j)} x_i\right),$$

  where the function bi2re transitions from the Galois field to real-values numbers, cf. Section 4.2.2.

Using this factorization, the iterative sum-product algorithm can be performed. The result of this process are the well-known estimates for the marginalization probabilities.

- As LDPC codes contain cycles for sure, the decoding graph is not a tree any more. For this reason, the stopping criterion described in Section 4.3 does not hold any longer. Keeping the algorithm running, even after all edges have passed information in every direction, turns out to provide for performance improvements and is common practice.

This leads us to a presentation of the decoding algorithm which follows the presentation by MacKay [Mac03, p. 560] closely. All necessary steps are given in Algorithm 4.2.

Algorithm 4.2 Belief-Propagation using the sum-product algorithm.

Input: $\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, optionally p^{ext}
Output: $\hat{\boldsymbol{x}}$, success_flag, p^{ext}
Function call: $[\hat{\boldsymbol{x}}, \text{success\_flag}, p^{\text{ext}}]$=bp_decode_2($\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, $\langle p^{\text{ext}} \rangle$)

1: **/* Initialize variable node probabilities */**
2: $p_i^{\text{int}} = \Pr(X_i = 0 \mid Y_i = y_i), i = 1, \ldots, n$
3: $p_i^{\text{app}} = p_i^{\text{int}}, i = 1, \ldots, n, \boldsymbol{p}^{\text{int/app}} = [p_1^{\text{int/app}}, \ldots, p_n^{\text{int/app}}]$
4: **if** p^{ext} **not defined then**
5: $\quad p_{ij}^{\text{ext}} = p_i^{\text{int}}, i = 1, \ldots, n, j \in \mathcal{N}(\mathrm{v}_i)$
6: **end if**
7: **while** $I < I_{\max}$ **and** $\lceil \boldsymbol{p}^{\text{app}} \rfloor \odot \boldsymbol{H}^{\mathsf{T}} \neq \boldsymbol{0}$ **do**
8: $\quad$ **/* Process information at the factor nodes */**
9: $\quad$ **for** $j = 1, \ldots, m$ **do**
10: $\quad\quad q_{jj'}^{\text{ext}} = \sum_{\boldsymbol{x} \in \text{hyp}(\mathcal{X}, \mathcal{N}(\mathrm{F}_j) \backslash j')} \text{bi2re}\left(1 \oplus (x_{j'} = 0) \oplus \bigoplus_{j'' \in \mathcal{N}(\mathrm{F}_j) \backslash j'} x_{j''}\right) \cdot \ldots$
11: $\quad\quad \ldots \prod_{j'' \in \mathcal{N}(\mathrm{F}_j) \backslash j'} (p_{j''j}^{\text{ext}})^{1 - x_{j''}} (1 - p_{j''j}^{\text{ext}})^{x_{j''}}$
12: $\quad\quad$ **/* Process information at the variable nodes */**
13: $\quad\quad p_i^{\text{app}} = \eta_i \cdot p_i^{\text{int}} \cdot \prod_{i' \in \mathcal{N}(\mathrm{v}_i)} q_{i'i}^{\text{ext}}$
14: $\quad\quad p_{ii'}^{\text{ext}} = \eta_{ii'} \cdot p_i^{\text{int}} \cdot \prod_{i'' \in \mathcal{N}(\mathrm{v}_i) \backslash i'} q_{i''i}^{\text{ext}}$
15: $\quad$ **end for**
16: $\quad I = I + 1$
17: **end while**
18: **/* Hard decision using a-posteriori info */**
19: $\hat{\boldsymbol{x}} = \lceil \boldsymbol{p}^{\text{app}} \rfloor$
20: success_flag $= \sum \text{bi2re}(\hat{\boldsymbol{x}} \odot \boldsymbol{H}^{\mathsf{T}})$

Some of the steps of Algorithm 4.2 require further explanation. The core of the algorithm are lines 8 and 9, which represent the equations for factor node updates, and lines 11 and 12, which represent the equations at the variable node updates. The function $\text{hyp}(\mathcal{X}, \mathcal{N}(\mathrm{F}_j) \backslash j')$ denotes all hypothetical vectors that are of interest for calculation of the formula in lines 8 and 9, i.e. it returns vectors which contain all possible patterns from alphabet $\mathcal{X}$ on the set of positions $\mathcal{N}(\mathrm{F}_j) \backslash j'$. In line 8, the argument of the function bi2re selects all valid codewords with $x_{j'} = 0$.

The product formula in line 9 of Algorithm 4.2 calculates the corresponding product patterns[2]. It is also easy to see that $1 - q_{jj'}^{\text{ext}}$ can be expressed by setting $x_{j'} = 1$, i.e.

$$1 - q_{jj'}^{\text{ext}} = \sum_{\boldsymbol{x} \in \text{hyp}(\mathcal{X}, \mathcal{N}(\text{F}_j) \backslash j')} \text{bi2re}\left(1 \oplus (x_{j'} = 1) \oplus \bigoplus_{j'' \in \mathcal{N}(\text{F}_j) \backslash j'} x_{j''}\right) \cdot \prod_{j'' \in \mathcal{N}(\text{F}_j) \backslash j'} (p_{j''j}^{\text{ext}})^{1 - x_{j''}} (1 - p_{j''j}^{\text{ext}})^{x_{j''}}. \tag{4.7}$$

Example 4.3.3 allows us to identify this formula as the Gallager formula.

Example 4.3.3 *Consider a factor node* F_j *with the variable nodes corresponding to* x_1*,* x_2*, and* x_3 *as its neighborhood. We are interested in calculating* q_{j3}^{ext}*. In a first step we set* $x_3 = 0$*. Identify* $j' = 3$ *and observe that* $j'' \in \{1, 2\}$*. Four hypothetical vectors are tested. The valid combinations for the remaining variables are* $[x_1, x_2] = [0, 0]$ *and* $[x_1, x_2] = [1, 1]$*. Hence, we obtain as a result of lines 8 and 9 of Algorithm 4.2*

$$q_{j3}^{\text{ext}} = p_{1j}^{\text{ext}} \cdot (1 - p_{1j}^{\text{ext}})^0 \cdot p_{2j}^{\text{ext}} \cdot (1 - p_{2j}^{\text{ext}})^0 + (p_{1j}^{\text{ext}})^0 \cdot (1 - p_{1j}^{\text{ext}}) \cdot (p_{2j}^{\text{ext}})^0 \cdot (1 - p_{2j}^{\text{ext}}).$$

Setting $x_3 = 1$*, the valid combinations for the remaining variables read* $[x_1, x_2] = [1, 0]$ *and* $[x_1, x_2] = [0, 1]$*. Consequently, we obtain from Equation (4.7) the result* $1 - q_{j3}^{\text{ext}} = (1 - p_{1j}^{\text{ext}}) \cdot p_{2j}^{\text{ext}} + p_{1j}^{\text{ext}} \cdot (1 - p_{2j}^{\text{ext}})$*. Subtraction of these terms leads to* $2q_{j3}^{\text{ext}} - 1 = (2p_{1j}^{\text{ext}} - 1) \cdot (2p_{2j}^{\text{ext}} - 1)$*.*

At the variable-node side, the calculation of p_i^{app} is a product which still involves an unknown factor η_i. This variable is chosen in such a way that p_i^{app} is indeed a probability, i.e. such that line 11 in Algorithm 4.2 as well as

$$1 - p_i^{\text{app}} = \eta_i \cdot (1 - p_i^{\text{int}}) \cdot \prod_{i' \in \mathcal{N}(\text{v}_i)} (1 - q_{i'i}^{\text{ext}})$$

holds. Thus, η_i needs to be set to

$$\eta_i = \frac{1}{p_i^{\text{int}} \cdot \prod\limits_{i' \in \mathcal{N}(\text{v}_i)} q_{i'i}^{\text{ext}} + (1 - p_i^{\text{int}}) \cdot \prod\limits_{i' \in \mathcal{N}(\text{v}_i)} (1 - q_{i'i}^{\text{ext}})},$$

and with this it becomes clear that η_i is just a variable which "disguises" the normalized multiplication (cf. Equation (3.14)) as a product. Similar conclusions can be made for $\eta_{ii'}$.

The *min-sum algorithm* is a low-complexity approximation of the sum-product algorithm and performs slightly worse [Ana01]. In this thesis, we only consider decoding by the sum-product algorithm and henceforth refer to it as belief-propagation decoding.

[2]MacKay [Mac03, p. 560] writes the formulas in lines 8 and 9 without using the additional product $p \cdot (1 - p)$ and thus as a direct product of probabilities. However, this requires to use both the probabilities for $X = 0$ and $X = 1$.

Chapter 5

Channel Codes - Basic Definitions and Realizations

This chapter provides basic definitions of coding theory as well as detailed information on the most important codes applied in this thesis. Also, the simulation setup used to generate the presented results is briefly summarized. In general, we restrict our attention to binary linear codes and distinguish between block codes and convolutional codes [LC04]. Considering the block codes, we focus on cyclic algebraic codes as well as LDPC codes. In this thesis, block codes are consistently decoded by means of the BP algorithm and the modifications of BP introduced in this work. As to convolutional codes, we state properties of the codes used in this thesis and explain the algorithms used for decoding.

5.1 Block Codes

It is assumed that the reader is familiar with block codes [MS77]. For conciseness we repeat properties that are of special interest for this thesis.

An $[n, k, d]$ linear block code $\mathcal{C}$, cf. Definition 2.3.1, can be unambiguously defined by a generator matrix $\boldsymbol{G}(\mathcal{C})$. If the connection to $\mathcal{C}$ is clear, we denote the generator matrix by $\boldsymbol{G}$.

Definition 5.1.1 *For an $(k \times n)$ generator matrix $\boldsymbol{G}$ of a binary linear $[n, k, d]$ code $\mathcal{C}$, $\mathcal{C} = \{\boldsymbol{u} \odot \boldsymbol{G} \mid \boldsymbol{u} \in (\mathbb{F}_2)^k\}$ holds. The rank of the matrix $\boldsymbol{G}$ equals the code dimension k.*

In a next step, we define the code dual to a given code $\mathcal{C}$ as well as the dual distance.

Definition 5.1.2 *A code $\mathcal{C}^\perp$ is dual to an existing code $\mathcal{C}$ if all its codewords $\boldsymbol{v}$ are orthogonal to any codeword in $\mathcal{C}$ [MS77], i.e.*

$$\mathcal{C}^\perp = \{\boldsymbol{v} \mid \boldsymbol{v} \odot (\boldsymbol{u} \odot \boldsymbol{G})^\mathsf{T} = \boldsymbol{0}\} \quad \forall \boldsymbol{u} \in \mathbb{F}_2^k.$$

The dual code $\mathcal{C}^\perp$ is a linear $[n, n-k, d^\perp]$ block code. Its minimum distance $d^\perp$ is also referred to as the dual distance of $\mathcal{C}$.

A generator matrix allows to construct all possible 2^k codewords as linear combinations of the k rows of $\boldsymbol{G}$. These rows can be interpreted as the *basis* of $\mathcal{C}$ and the null space (kernel) of the dual code $\mathcal{C}^{\perp}$ [Wei02]. Besides Definition 5.1.1, a code can be defined in an unambiguous way by means of its null space. The rows of a parity-check matrix $\boldsymbol{H}(\mathcal{C})$ represent one basis for this space. We denote the parity-check matrix by $\boldsymbol{H}$ if the connection to $\mathcal{C}$ is clear.

Definition 5.1.3 *For an $(n-k \times n)$ parity-check matrix $\boldsymbol{H}$ of a binary linear $[n,k,d]$ code $\mathcal{C}$, $\boldsymbol{x} \odot \boldsymbol{H}^{\mathsf{T}} = \boldsymbol{0}$, $\forall \boldsymbol{x} \in \mathcal{C}$, holds. Furthermore, the rank of $\boldsymbol{H}$ needs to equal the co-dimension of the code, $n-k$.*

Note that parity-check matrices with more than $n-k$ rows are quite common in the field of iterative decoding. We refer to these matrices as *redundant* matrices and denote their number of rows by m. As all rows are assumed to differ from each other, $m \leq 2^{n-k}$ holds. Chapter 6 and Chapter 7 study this topic. For non-redundant matrices it is easy to see that

$$\boldsymbol{G}(\mathcal{C}) = \boldsymbol{H}(\mathcal{C}^{\perp}),$$

and since $\mathcal{C}^{\perp\perp} = \mathcal{C}$

$$\boldsymbol{G}(\mathcal{C}^{\perp}) = \boldsymbol{H}(\mathcal{C}).$$

Obviously, the generator matrix and parity-check matrix for a given linear code $\mathcal{C}$ are not unique. Actually, any k linearly independent codewords of the code and any $n-k$ linearly independent codewords of the dual code describe a generator matrix and parity-check matrix [LC04, p. 67], respectively.

All possible representations are equivalent if an optimal decoding algorithm is used. Given a received word $\boldsymbol{y}$, the maximum a-posteriori (MAP) algorithm performs an optimal codeword estimation and finds

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in \mathcal{C}}{\operatorname{argmax}} \Pr\left([X_1, \ldots, X_n] = [x_1, \ldots, x_n] \mid [Y_1, \ldots, Y_n] = [y_1, \ldots, y_n]\right). \tag{5.1}$$

The maximum-likelihood (ML) algorithm can be deduced from the previous result by means of the theorem of Bayes,

$$\Pr\left([X_1, \ldots, X_n] = [x_1, \ldots, x_n] \mid [Y_1, \ldots, Y_n] = [y_1, \ldots, y_n]\right) =$$
$$\frac{\Pr\left([Y_1, \ldots, Y_n] = [y_1, \ldots, y_n] \mid [X_1, \ldots, X_n] = [x_1, \ldots, x_n]\right) \Pr\left([X_1, \ldots, X_n] = [x_1, \ldots, x_n]\right)}{\Pr\left([Y_1, \ldots, Y_n] = [y_1, \ldots, y_n]\right)},$$

and considers the fact that $\Pr\left([Y_1, \ldots, Y_n] = [y_1, \ldots, y_n]\right)$ is insignificant for the maximization in Equation (5.1). Henceforth, a uniform probability distribution for the encoded words $\boldsymbol{x}$ will be assumed. The ML algorithm uses the decision rule

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\in\mathcal{C}}{\operatorname{argmax}} \Pr\left([Y_1, \ldots, Y_n] = [y_1, \ldots, y_n] \mid [X_1, \ldots, X_n] = [x_1, \ldots, x_n]\right). \tag{5.2}$$

The decoding performance is equivalent to that of the MAP decoder. Both algorithms can use any valid representation of $\boldsymbol{G}$ and $\boldsymbol{H}$ for decoding.

The situation is different for the *suboptimal* belief-propagation decoder, introduced in Chapter 4. The parity-check matrix directly corresponds to the Tanner graph introduced in Chapter 4 and the proper choice of this matrix is essential in order to obtain a good decoding performance. The sparseness of the matrix is one of the most important parameters when using this algorithm.

5.1.1 Cyclic Codes

Cyclic codes are linear block codes with special properties. Many of these characteristics lead to advantages in the decoding process. We start this section with the definition of a cyclic code.

Definition 5.1.4 *([MS77]) Let $\mathcal{C}$ be an $[n, k, d]$ binary linear code. A code is called cyclic iff any cyclic shift of a codeword $\boldsymbol{x} \in \mathcal{C}$ is also a codeword.*

Closely connected to the definition of cyclicality is the following remark.

Remark 5.1.5 *A parity-check matrix of $\mathcal{C}$ is said to be of cyclic form if it consists of $n - k \leq m \leq n$ consecutive cyclic shifts of one suitable codeword of $\mathcal{C}^{\perp}$. In this context, the qualifier consecutive expresses that the $i+1$-th row, $1 \leq i \leq m-1$ is the cyclic right shift by one position of the i-th row. For $m = n$, these matrices are also referred to as circulant matrices [Wei02].*

A (binary linear) cyclic code has at least one parity-check matrix of cyclic form. A code can have a parity-check matrix of cyclic form although it is not a cyclic code. These types of codes are not considered in this work.

One way to characterize a binary cyclic code completely is by means of a monic parity-check polynomial $\mathrm{h}(x) = h_0 + h_1 x + h_2 x^2 + \cdots + h_{k-1} x^{k-1} + 1x^k$ of degree k, for which $h_i \in \mathbb{F}_2$, $\forall\, i \in \{0, \ldots, k-1\}$. We construct a cyclic parity-check matrix $\boldsymbol{H}$ of the code by setting the entry in the i-th row and j-th column to

$$H[i, j] = \left.\frac{h^{(k-j+i \bmod n)}(x)}{(k-j+i \bmod n)!}\right|_{x=0},$$

where $h^{(j)}(x)$ denotes the j-th derivative of h(x) for x, $1 \leq i \leq m$, $1 \leq j \leq n$ and mod is the standard modulo function[1]. This leads to a parity-check matrix of the form

$$\boldsymbol{H} = \begin{bmatrix} 1 & h_{k-1} & \cdots & h_1 & h_0 & 0 & \dots & 0 \\ & \ddots & \ddots & & \ddots & & & \\ & & \ddots & \ddots & & \ddots & & \\ 0 & \dots & 0 & 1 & h_{k-1} & \cdots & h_1 & h_0 \end{bmatrix}.$$

In the upcoming investigations on cyclic codes we are interested in partitioning the set of codewords of the dual code. The partitioning which unites only words that are cyclic shifts of each other turns out to be very advantageous. In this context, we define a cyclic orbit as a set of codewords, all of which are cyclic shifts of each other.

Definition 5.1.6 *[HMLH08] Let $\mathcal{C}$ be a cyclic $[n, k, d]$ linear code. Partition the set of codewords of the dual code $\mathcal{C}^\perp$ into cyclic orbits. More precisely, let $\mathcal{C}^\perp$ be the union of disjoint sets $\bigcup_{\boldsymbol{x}} G_{\boldsymbol{x}}$ in which $\boldsymbol{x} \in \mathcal{C}^\perp$, $G_{\boldsymbol{x}} = \{\pi^i \, \boldsymbol{x},\ i = 1, \dots, n\}$, and where π denotes the cyclic shift permutation to the right. One designated element of $G_{\boldsymbol{x}}$ is henceforth referred to as the cyclic orbit generator (*COG*) of $G_{\boldsymbol{x}}$.*

Note that in general the first row of a redundant parity-check matrix does not necessarily have to be defined by a parity-check polynomial. The first row can be any COG which has the property that a sufficient number of its cyclic shifts generates a basis for the dual code.

One very prominent class of cyclic codes is the class of 2-ary Bose-Chaudhuri-Hocquenghem (BCH) codes [Hoc59, BRC60] of length $2^b - 1$, $b \in \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers[2]. These codes have a rich mathematical structure and are used at several places in this thesis. Another prominent code is the Golay code. In this thesis, we use the important binary Golay code of length 23 [Gol49]. Furthermore, we discuss results of extended cyclic codes, which are derived from cyclic codes by adding an additional parity bit. An extended cyclic code has the same dimension as the underlying cyclic code, but the length and the minimum distance are increased by one [MS77, p. 27]. The $[24, 12, 8]$ extended-Golay code [Gol49] is frequently applied in this thesis.

5.1.2 LDPC Codes

An LDPC code is a linear block code with the key property that it is defined by a sparse parity-check matrix.

Definition 5.1.7 *A matrix of size $(n-k) \times n$ is considered to be sparse if the number of 1-entries in a full-rank parity-check of $n - k$ rows grows linearly with n.*

[1] We assume that the modulus n is a non-negative integer.

[2] We assume that the set $\mathbb{N}$ does not include the number zero.

We will observe shortly that matrices realizing a *degree distribution* (still to be defined) are sparse. LDPC codes are known to provide good error-prevention and error-correction performance for a wide class of rates and code lengths. They also show a good minimum distance [Mac03, p. 557] in general. Here, the term 'good minimum distance' means that the ratio d/n tends to a positive constant [Mac03, p. 207] if codes of different length but with the same degree distribution are considered. However, comparisons of algebraic codes and constructed LDPC codes of length $n \leq 1000$ and rate $R = 1/2$ have shown that algebraic codes usually obtain a higher minimum distance when codes of equal length are compared to each other. LDPC codes are capable of preventing way more errors than half the minimum distance and often even exceed the minimum distance [Mac03, p. 207].

Degree Distributions and Regular Codes

Degree distributions specify how edges of a Tanner graph are connected to the corresponding nodes. We distinguish between degree distributions from the node perspective and edge perspective.

Definition 5.1.8 *Given a parity-check matrix $\boldsymbol{H}$, the variable Λ_i, $1 \leq i \leq l_{\max}$ specifies the number of variable nodes* v *that have i edges connected in the Tanner graph corresponding to $\boldsymbol{H}$. Equivalently, P_j, $1 \leq j \leq r_{\max}$ specifies the number of check nodes* c *that have j edges connected in that Tanner graph. The variables $l_{\max}$ and $r_{\max}$ denote the maximum degree on the left-hand side and right-hand side, respectively.*

Accordingly, we define the *degree distributions from the node perspective* as

$$\Lambda(x) = \sum_{i=1}^{l_{\max}} \Lambda_i x^i, \qquad \mathrm{P}(x) = \sum_{i=1}^{r_{\max}} \mathrm{P}_i x^i.$$

Naturally, $\Lambda(1) = n$ and $\mathrm{P}(1) = m$ has to hold. It is often more convenient to use the *normalized* degree distribution from the node perspective,

$$\mathrm{L}(x) = \frac{\Lambda(x)}{\Lambda(1)}, \qquad \mathrm{R}(x) = \frac{\mathrm{P}(x)}{\mathrm{P}(1)}.$$

Note that the notation Λ and P, as well as L and R are chosen to remind of the left-hand side and the right-hand side of the Tanner graph, cf. Figure 4.1. Often, it is of interest to identify the (normalized) fraction of edges that emanate from variable nodes with given degree. This requires to multiply each fraction with the current degree, what is realized by deriving the polynomial for x,

$$\lambda(x) = \frac{\Lambda'(x)}{\Lambda'(1)} = \frac{\mathrm{L}'(x)}{\mathrm{L}'(1)}, \qquad \rho(x) = \frac{\mathrm{P}'(x)}{\mathrm{P}'(1)} = \frac{\mathrm{R}'(x)}{\mathrm{R}'(1)}.$$

The general form of these distributions reads $\lambda(x) = \sum_{i=1}^{l_{\max}} \lambda_i x^{i-1}$, and $\rho(x) = \sum_{i=1}^{r_{\max}} \rho_i x^{i-1}$. Also, note that $\Lambda'(1) = \mathrm{P}'(1)$ equals the number of total edges in the graph.

Let us consider a set of j codes of different lengths $n_1, \ldots, n_j$, all of them realizing the same normalized degree distribution. Then, the number of edges in the Tanner graphs of these codes is given as $\lambda(1) \cdot n_\iota$, $\iota \in \{1, \ldots, j\}$. This provides us with the fundamental insight that parity-check matrices following a degree distribution can be identified as sparse matrices, cf. Definition 5.1.7.

Regular codes[3] follow a special normalized degree distribution with $\mathrm{L}_i = \mathrm{R}_j = 1$, where i and j denote one fixed number each. *Check-regular* codes satisfy $\mathrm{R}_j = 1$ for a given choice of j. Let us next specify what concentration in the context of a Tanner graph means.

Definition 5.1.9 ([HEA05]) *A graph is said to be concentrated if only nodes of two consecutive non-zero degrees and no other degrees exist.*

It will be to observe that we are especially interested in code representations which are concentrated on the check node side.

It is known that irregular codes achieve better performance results than their regular counterparts [RU08]. Reasons for this behavior include the fact that variable nodes of higher degrees tend to converge within a few number of decoder iterations [Lec03] and support variable nodes of lower degree to converge. Contrary, the solemn use of high-degree variables would result in a comparatively dense parity-check matrix. It is of great interest to determine degree distributions that compare favorably, what is accomplished by *density evolution* [RSU01]. Optimized degree distributions for any rate can be obtained from the web database [Urb] provided by R. Urbanke. For the binary erasure channel, density evolution is equivalent to an analysis [HDH05a] by means of EXIT charts [tB99].

Local Girth Distribution

The (normalized) local girth distribution [MB01a] specifies the fraction of variable nodes with a local girth (cf. Definition 4.2.2) of a given value and is specified by

$$\mathrm{lgd}_{\mathcal{G}}(x) = \sum_{j=\mathrm{gi}(\mathcal{G})}^{j_{\max}} \frac{|\{i \mid \mathrm{lgi}(\mathcal{G}, \mathrm{v}_i) = j\}|}{n} x^j,$$

where the fraction $\frac{|\{i|\mathrm{lgi}(\mathcal{G},\mathrm{v}_i)=j\}|}{n}$ denotes the ratio of variable nodes having a local girth of length j. The variables $\mathcal{G}$ and $\mathrm{lgi}(\mathcal{G}, \mathrm{v}_i)$ are defined on page 21 and 26, respectively. The variable $j_{\max}$ represents the highest local girth in the considered Tanner graph.

[3]The denomination 'regular code' is misleading, as the regularity is not a property of the code but the parity-check matrix used to represent it in the decoder. The author objects the terminology 'regular code' but introduces it as it is widely used in literature [HEA05, RU08].

Constructing LDPC Codes

We present a short outline on how to construct LDPC codes with good error prevention performance. Robert Gallager constructed parity-check matrices of low weight in a random way [Gal63]. In his thesis, Gallager experimented with codes of short to moderate length ($n \leq 1008$) as the capabilities for computer simulations were limited at that time. For the AWGN channel, he reported a bit error rate (BER), to be defined shortly, not lower than 10^{-4}. This error rate was obtained with rate-1/2 codes of length $n = 504$ at $10\log_{10}(E_\mathrm{b}/N_0) = 3\,\mathrm{dB}$. Reasons for this limited performance are the code length, the regular degree distribution, as well as the random construction process. Early papers after the 'rediscovery' of LDPC codes [MN96] presented results for LDPC codes the matrices of which follow irregular degree distributions [LMSS98] or are constructed by means of more sophisticated random algorithms [Mac99]. Some of these algorithms applied a post-processing step to the matrix to avoid local cycles of length 4 as good as possible. Density evolution [RSU01] has shown that there exist irregular degree distributions which lead to very good performance thresholds, i.e. very good performance results for codes with a length approaching infinity. Optimized values for almost any rate and any maximum degree can be found at [Urb].

These papers show that there exists room for improving the results by Gallager, but they all rely on long codes. These presented results have in common that they are connected to a high code length. For short-to-moderate code lengths, random construction processes cannot be used, as short cycles occur with high probability. As a consequence, the probability of a code realization with poor decoding performance is high [HEA05, DPT+02]. In order to obtain codes with moderate block length and good decoding performance, we construct the parity-check matrix by means of an algorithm. To this end, we choose a construction algorithm which creates parity-check matrices of high-girth which realize a given degree distribution. The most prominent algorithm of this type is the progressive edge-growth (PEG) algorithm [HEA05] and its variations [XB04]. To the best knowledge of the author, the greedy PEG algorithm is the construction algorithm leading with the best error-prevention performance on the AWGN channel when a codelength of $n \leq 1000$ is desired.

As both density evolution and the PEG algorithm lead to promising results, it is tempting to combine both algorithms to construct well performing codes. Yet, this cannot be accomplished in a straightforward way. It has been observed that degree distributions with desirable thresholds obtain a quite poor performance when being used for constructing codes of moderate lengths [HDH05b]. Reasons for this include the high degrees and poor girth profiles associated with these degree distributions as well as the consequential high density of the constructed matrices.

Let us now focus on channel codes of rate 1/2. These codes are the most challenging codes for decoding as they show both a high dimension and a high co-dimension. We use the PEG algorithm to construct codes of length $n \leq 1000$ realizing different degree distributions: two optimized distributions found at [Urb], one of them with a maximum variable node degree of

$l_{\max} = 100$, [4] the other one with $l_{\max} = 12$. Also, a regular code is considered. The distributions read

$$\begin{aligned}
\mathrm{L}_1(x) &= 0.46477793676039x^2 + 0.29561427779418x^3 + 0.0065324111386271x^5 \\
&\quad +0.015399739471419x^6 + 0.14650049481207x^7 + 0.0026968764628759x^{14} \\
&\quad +0.0083764755185871x^{17} + 0.02790349464248x^{19} + 0.0021453279419301x^{27} \\
&\quad +0.00091905872079926x^{28} + 0.01509971418948x^{29} + 0.0016761759341078x^{39} \\
&\quad +0.012358016613056x^{100}, \\
\mathrm{R}_1(x) &= x^{11},
\end{aligned}$$

$$\begin{aligned}
\mathrm{L}_2(x) &= x^3, \\
\mathrm{R}_2(x) &= x^6,
\end{aligned}$$

$$\begin{aligned}
\mathrm{L}_3(x) &= 0.50438655583665x^2 + 0.29557605288481x^3 + 0.05726340795648x^5 \\
&\quad +0.03626021937498x^6 + 0.0049622080898005x^7 + 0.029234477635091x^9 \\
&\quad +0.065031247693527x^{11} + 0.0072858305286566x^{12}, \\
\mathrm{R}_3(x) &= x^7.
\end{aligned}$$

Ensembles and Sets of PEG-Code Ensembles

In the following, we introduce the idea of an ensemble and generalize it in two steps. First, we define a set of ensembles by allowing different code lengths. As a second step, we allow only code realizations constructed by the PEG algorithm. We refer to these entities as sets of PEG-ensembles. With this, we provide a proper description of the codes used in this thesis. A (standard) LDPC ensemble is defined as a set of graphs realizing a degree distribution pair $\Lambda(x)$, $\mathrm{R}(x)$, together with a probability distribution for the graphs. To describe this rigorously, we first introduce the notation required to describe a permutation.

Definition 5.1.10 *We write $\zeta(\boldsymbol{x})$ if we mean that some permutation ζ is applied to the vector $\boldsymbol{x}$. Also, we use the notation $\zeta : i \to i'$, $i, i' \in \{1, \ldots, n\}$ for a function that maps the indices of the codeword.*

Let us now proceed by introducing the ensemble.

[4] At the time this thesis was compiled, this distribution had the best threshold from all distributions of rate 1/2 in [Urb] (Gap to Shannon Limit: 0.024 dB).

Definition 5.1.11 *([RU08, p. 79]) Given a degree distribution pair* $\Lambda(x)$, $\mathrm{R}(x)$, *define an LDPC ensemble of bipartite graphs (code realizations) the following way. Each graph in the LDPC ensemble has* $\Lambda(1)$ *variable nodes and* $\mathrm{R}(1)$ *check nodes,* Λ_i *variable nodes and* R_i *check nodes of degree* i. *A node of degree* i *has* i *sockets from which* i *edges emanate. Label the sockets on each side in some arbitrary but fixed way. Let* $\zeta(\cdot)$ *be a permutation on the socket labeling vector. Associate to* $\zeta(\cdot)$ *a bipartite graph by connecting a variable node and a check node such that the indices of these two nodes are connected to each other by* ζ. *Finally, we define a probability distribution over the set of graphs by placing the uniform probability distribution over the set of permutations.*

Note that we use the more convenient term ensemble instead of LDPC ensemble if the connection to LDPC codes is clear. Definition 5.1.11 states that in an ensemble the codelength n is strictly fixed. We extend the idea of the ensemble to a set of ensembles. Restricting each ensemble in this set to code realizations that correspond to a PEG-constructed code leads us to the set of PEG-ensembles $\mathcal{U}$. All members of $\mathcal{U}$ are ensembles of PEG codes and follow the same degree distribution.

Properties of the Codes Constructed in this Thesis

Consider again the distributions $\mathrm{L}_1(x)$, $\mathrm{R}_1(x)$ to $\mathrm{L}_3(x)$, $\mathrm{R}_3(x)$. The corresponding sets of PEG-code ensembles are denoted by $\mathcal{U}_1$ to $\mathcal{U}_3$, where we restrict our attention to the code lengths $n = 50$, $n = 100, 200, \ldots, 1000$. Table 5.1 shows the thresholds $\psi_{\mathrm{L}_i(x),\mathrm{R}_i(x)}$, $i = 1, 2, 3$, (found by density evolution, assuming a real-valued channel and $R = 1/2$ for calculation of $10\log_{10}(E_\mathrm{b}/N_0)$) for the degree distribution. For comparison reasons we state that the capacity limit for codes of rate 1/2 (BPSK signaling and AWGN channel) reads 0.18706 dB. Note that only the variable node degree distributions need to be specified for the PEG algorithm, as it aims at constructing graphs which are concentrated on the check node side, cf. Definition 5.1.9.

Set $\mathcal{U}_i$ of PEG-code ensembles	Threshold $\psi_{\mathrm{L}_i(x),\mathrm{R}_i(x)}$ in $10 \cdot \log_{10}(E_\mathrm{b}/N_0)$
$\mathcal{U}_1$	$\psi_{\mathrm{L}_1(x),\mathrm{R}_1(x)} = 0.21076\,\mathrm{dB}$
$\mathcal{U}_2$	$\psi_{\mathrm{L}_2(x),\mathrm{R}_2(x)} = 1.10156\,\mathrm{dB}$
$\mathcal{U}_3$	$\psi_{\mathrm{L}_3(x),\mathrm{R}_3(x)} = 0.39610\,\mathrm{dB}$

Table 5.1: Thresholds for the considered degree distribution.

Figure 5.1 shows the required signal-to-noise ratio to obtain FER $\leq 10^{-3}$ for the different PEG-code ensembles in the considered set with $n \leq 1000$. In order to compare the different distributions, we use the frame error rate (FER). This rate specifies the ratio of unsuccessfully recovered frames after channel decoding and is introduced rigorously on page 46. As the PEG algorithm uses random seeds in order to construct a specific code, we simulate 12 different seeds for each code length and report that the decoding performance of the constructed codes is equal for the frame error rates of interest. This is in accord with the *concentration theorem*,

which essentially states that it is very probable that codes from one ensemble have comparable performance [RU08, p. 85]

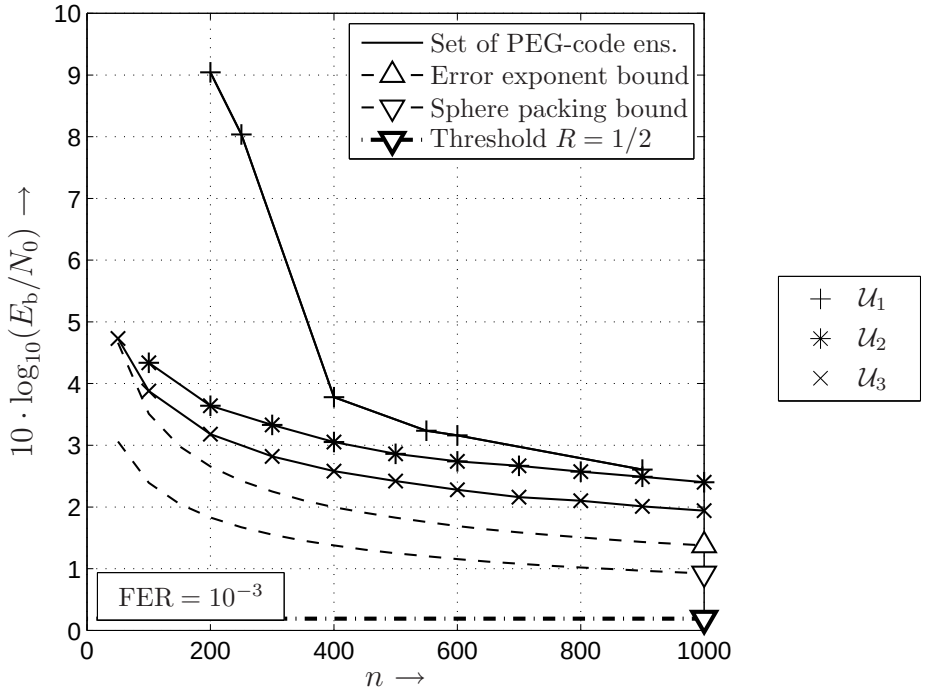


Figure 5.1: Required SNR to obtain FER $= 10^{-3}$ with different sets of PEG-code ensembles ($R = 1/2$, 50 iterations). For comparison, several performance bounds for block coding schemes, to be introduced in Section 5.1.4, are included.

Comparison of Table 5.1 and Figure 5.1 shows that within the considered range of code lengths, the threshold is not a valid measure for good error prevention performance. Instead, simulations are required to identify a favorable degree distribution. We report that the set of ensembles $\mathcal{U}_3$ obtained the best performance within the range $50 \leq n \leq 1000$. Additionally, bounds from information theory, which will be introduced shortly, are shown to assess the results in a more general manner. In the following, we provide detailed information on codes from the set of PEG-code ensembles $\mathcal{U}_3$ with $50 \leq n \leq 1000$.

Table 5.2 shows average values for codes from the set of ensembles $\mathcal{U}_3$. The expectation is taken separately for each codelength and runs over selected codes[5] in $\mathcal{U}_3$, each of them represented by the graph constructed by the PEG algorithm. We denote the expectation by the operator $\mathrm{E}\{\cdot\}$ and use the index to indicate the variable in the averaging process. Table 5.2 shows the average local girth distribution together with an average upper bound on the minimum distance $\overline{\mathrm{B}}(d)$, estimated by the methods provided in [HFE04]. It is to observe that these PEG codes are a family with a *bad* distance [Mac03, Ch. 13].

[5] 12 codes were deployed for each code length.

n	$\mathrm{E}_{\mathcal{C}\in\mathcal{U}_3\|n}\{\mathrm{lgd}_{\mathcal{G}}(x)\}$	$\mathrm{E}_{\mathcal{C}\in\mathcal{U}_3\|n}\{\overline{\mathrm{B}}(d)\}$
50	$0.82 \cdot x^4 + 0.18 \cdot x^6$	5
100	$0.42 \cdot x^4 + 0.58 \cdot x^6$	6
200	$0.045 \cdot x^4 + 0.955 \cdot x^6$	8
300	$0.9967 \cdot x^6 + 0.0033 \cdot x^8$	9
400	$0.9850 \cdot x^6 + 0.015 \cdot x^8$	11
500	$0.9320 \cdot x^6 + 0.068 \cdot x^8$	9
600	$0.9133 \cdot x^6 + 0.0867 \cdot x^8$	10
700	$0.8429 \cdot x^6 + 0.1571 \cdot x^8$	12
800	$0.8113 \cdot x^6 + 0.1888 \cdot x^8$	9
900	$0.7489 \cdot x^6 + 0.2511 \cdot x^8$	12
1000	$0.669 \cdot x^6 + 0.331 \cdot x^8$	14

Table 5.2: Investigated PEG codes from the set of ensembles $\mathcal{U}_3$. Shown are the average local girth distribution and average estimated upper bound on the minimum distance, $\overline{\mathrm{B}}(d)$, for different code lengths n. When the code length n is fixed, the average is taken over selected codes from $\mathcal{U}_3$.

n	w	R
576	24	1/2
672	28	1/2
768	32	1/2
864	36	1/2
960	40	1/2

Table 5.3: Length, expansion factor, and rate of the WiMAX codes used in this thesis.

LDPC Codes from Industry Standards

Besides the PEG-constructed LDPC codes, we consider the performance of codes from communications standards, e.g. standards by the IEEE or the European Telecommunications Standards Institute (ETSI). Recently it has been to observe that communications standards apply more and more codes of short or moderate length. While the DVB-S2 standard [JTC05] still uses codes of length $n = 64800$, the IEEE 802.3an communications standard [IEE06] already uses channel codes of length $n = 2048$. We focus on LDPC codes of rate 1/2 which are used in the IEEE 802.16e (Worldwide Interoperability for Microwave Access, WiMAX) standard [IEE05, p. 626]. The reason for this choice is, besides the rate of 1/2, the moderate length of these codes. The lengths start at $n = 576$ and increase in steps of 96 up to $n = 2304$. To each length corresponds an *expansion factor*, denoted by w. The denomination expansion factor is due to its involvement in the creation of the parity-check matrix of the code, detailed in Section 7.7.2. From the set of standardized lengths and expansion factor, we use the subset listed in Table 5.3. A detailed description of the parity-check matrices is given in Section 7.7.2.

5.1.3 Simulation Details for Block Codes

This section deals with technical details describing how simulation results of the iterative decoding schemes were generated. To be precise, we consider the estimation of all relevant parameters and settings which are required to estimate the bit error rate (BER) and the frame error rate (FER). This detailed description at hand, readers are able to reproduce the results discussed in this thesis.

Linear block codes require a generator matrix $\boldsymbol{G}$, which transforms each block of source symbols $\boldsymbol{u}$ into a block of encoded symbols $\boldsymbol{x}$, cf. Section 2.3. Only the constant transmission of the all-zero codeword supersedes the generator matrix. This leads to valid results in general [Gal63, Ch. 6] as linear codes are used but is neither recommended nor conducted by the author.

It is a desirable *encoding property* if the source symbols can be found within the block of encoded symbols. This approach is referred to as *systematic encoding*. Usually, the source symbols are located in a block at the beginning of the block of encoded symbols. However, it is not necessary to choose this position or to place them in a contiguous block. Denoting the set of positions by $\mathcal{P}_{\text{syst}}$, the estimation of $\hat{\boldsymbol{u}}$ from $\hat{\boldsymbol{x}}$ is done in a straightforward manner by

$$\hat{\boldsymbol{u}} = \hat{\boldsymbol{x}}(\mathcal{P}_{\text{syst}}),$$

where we use the notation $\boldsymbol{x}(\mathcal{P}_{\text{syst}})$, cf. Section 4.3, to denote all elements of $\boldsymbol{x}$ indexed by the set $\mathcal{P}_{\text{syst}}$. In order to get desirable results, it is advisable to use systematic encoding. This is motivated by the fact that all other choices require post-processing by an inverse of the generator matrix. If not stated differently, results in this thesis are generated by systematic encoding.

Let us now elaborate on the consequences to the performance for the case that non-systematic encoding is applied. There exists a set of positions $\mathcal{P}_{\text{inv}}$ for which the *restriction* (defined subsequently) of the matrix $\boldsymbol{G}$ has full rank and is thus invertible.

Definition 5.1.12 *Let the columns of $\boldsymbol{G}$ be indexed by $\mathcal{P} = \{1, \ldots, n\}$, cf. Section 4.3. For a set $\mathcal{P}' \subseteq \mathcal{P}$, $|\mathcal{P}'| \geq 1$, the restriction of $\boldsymbol{G}$ to $\mathcal{P}'$ is the matrix that consists of the set of columns of $\boldsymbol{G}$ indexed by the elements of $\mathcal{P}'$, and is denoted by $\boldsymbol{G}(:, \mathcal{P}')$.*

The choice of $\mathcal{P}_{\text{inv}}$ is usually not unique. Independent of this choice, the inverse of the restriction of $\boldsymbol{G}$, $\text{inv}(\boldsymbol{G}(:, \mathcal{P}_{\text{inv}}))$, is a dense matrix. The estimation of the source symbols reads

$$\hat{\boldsymbol{u}} = \hat{\boldsymbol{x}}(\mathcal{P}_{\text{inv}}) \odot \text{inv}(\boldsymbol{G}(:, \mathcal{P}_{\text{inv}})).$$

Especially interesting is the case when signaling over the BEC. Here, residual erasures have to be removed by guessing the transmitted information at erased positions within $\hat{\boldsymbol{x}}(\mathcal{P}_{\text{inv}})$. As we restrict ourselves to binary codes, this is done correctly with a probability of 1/2. The post-processing with $\text{inv}(\boldsymbol{G}(:, \mathcal{P}_{\text{inv}}))$ leads either to the correct block of source symbols or to a block which differs from $\boldsymbol{u}$ in a significant amount of positions.

We define the bit error rate as

$$\mathrm{BER} = \frac{1}{k}\sum_{i=1}^{k} \Pr\left((U_i = u_i) \neq (\hat{U}_i = \hat{u}_i)\right),$$

and the frame error rate as

$$\mathrm{FER} = \Pr\left(([U_1, \ldots, U_k] = [u_1, \ldots, u_k]) \neq ([\hat{U}_1, \ldots, \hat{U}_k] = [\hat{u}_1, \ldots, \hat{u}_k])\right).$$

It is observed that for non-systematic encoding there exists a weak coherence between BER and FER which reads $\mathrm{BER} \lessapprox \mathrm{FER}/2$. The factor $1/2$ originates from the fact that the matrix $\mathrm{inv}(\boldsymbol{G}(:, \mathcal{P}_{\mathrm{inv}}))$ has a high-density (≈ 0.5) and post-processing with this matrix leads to estimated codewords with a high error density. The exact relation depends on the choice of both $\boldsymbol{G}$ and the set $\mathcal{P}_{\mathrm{inv}}$, which are not unique. Whenever simulation results with non-systematic encoding are shown, it is assumed that a dense representation of $\mathrm{inv}(\boldsymbol{G}(:, \mathcal{P}_{\mathrm{inv}}))$ has been used. The reason for this proceeding is to provide the reader with the two extreme cases for the inversion of the encoding process.

Remark 5.1.13 *Assume that systematic encoding is used. Simulation results for all codes decoded by BP have shown that the measured* BER *and* FER *do not differ substantially, if being defined as*

$$\mathrm{BER} = \frac{1}{n}\sum_{i=1}^{n} \Pr\left((X_i = x_i) \neq (\hat{X}_i = \hat{x}_i)\right),$$

and

$$\mathrm{FER} = \Pr\left(([\hat{X}_1, \ldots, \hat{X}_n] = [\hat{x}_1, \ldots, \hat{x}_n]) \neq ([X_1, \ldots, X_n] = [x_1, \ldots, x_n])\right),$$

respectively. Note that for the BEC case, one needs to eliminate erasures by a guessing routine before comparing $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$. This observation allows to state that the performance of the considered schemes is independent of the choice of $\mathcal{P}_{\mathrm{syst}}$ when systematic encoding is used.

5.1.4 Theoretical Limits for Block Codes

In order to assess the results presented throughout this thesis in a more general manner, we discuss theoretical bounds on the power efficiency of a channel coding scheme. As these bounds are well known from the field of information theory, we restrict ourselves to a brief discussion. We use the *union bound* as an upper bound on the decoding performance of a specific code decoded by the ML decoder. Simulating the performance of an ML decoder is often computationally infeasible but the weight enumerator of the code is available or can be approximated. The union bound for BPSK transmission over the AWGN channel using linear block codes is of the form

$$\mathrm{BER} \leq \frac{1}{n} \sum_{i=d}^{n} A_i \cdot i \cdot \mathrm{Q}\left(\sqrt{2\frac{k}{n} \cdot i \cdot \frac{E_\mathrm{b}}{N_0}}\right) \tag{5.3}$$

and

$$\mathrm{FER} \leq \sum_{i=d}^{n} A_i \cdot \mathrm{Q}\left(\sqrt{2\frac{k}{n} \cdot i \cdot \frac{E_\mathrm{b}}{N_0}}\right), \tag{5.4}$$

where $\mathrm{A}(x) = \sum_{i=0}^{n} A_i \cdot x^i$ [MS77, Ch. 2] denotes the weight enumerator of the code, and the variable A_i is defined as the number of codewords of weight i. The latter variable is either known from tables, provided e.g. by [MS77], or computer search strategies. The function

$$\mathrm{Q}(i) = \frac{1}{2\pi} \int_i^\infty \mathrm{e}^{-j^2/2}\,\mathrm{d}j, i \in \mathbb{R},$$

denotes the complementary Gaussian error integral. Within the error rates of interest (BER $\leq 10^{-4}$, FER $\leq 10^{-3}$), the union bound is known to be tight.

The *Gallager bound*[6] [Gal68] considers an ensemble of all block codes of rate R and length n and provides an upper bound on the expected FER. Note that in this context, the ensemble is not restricted to (PEG-)codes realizing a specific degree distribution. The ensemble contains all possible codes of a specified code length n and the expectation is taken over this ensemble. As this includes codes with very poor performance, realizations outperforming this bound are possible. The currently known block codes of rate 1/2 do not outperform this bound in general, thus it flags aspired values on the power efficiency. This bound reads

$$\mathrm{E}_\mathcal{C}\{\mathrm{FER}\} \leq \mathrm{e}^{-n\cdot\left(\mathrm{E}_0\left(\rho_\mathrm{D},\Pr([X_1,\ldots,X_{|\mathcal{X}|}]=[x_1,\ldots,x_{|\mathcal{X}|}]),10\cdot\log_{10}(E_\mathrm{b}/N_0)\right)-\rho_\mathrm{D}\cdot R\right)}, \tag{5.5}$$

where ρ_D is a design parameter and $\Pr([X_1,\ldots,X_{|\mathcal{X}|}] = [x_1,\ldots,x_{|\mathcal{X}|}])$ denotes the vector of probabilities of possible channel inputs. Maximization over the design parameter ρ_D and the vector $\Pr([X_1,\ldots,X_n] = [x_1,\ldots,x_n])$ leads to

$$\mathrm{E}_\mathcal{C}\{\mathrm{FER}\} \leq \mathrm{e}^{(-n\cdot\mathrm{E}_r(R))}, \tag{5.6}$$

where the random coding exponent

$$\mathrm{E}_\mathrm{r}(R) = \max_{0\leq\rho_\mathrm{D}\leq 1} \max_{\Pr([x_1,\ldots,x_{|\mathcal{X}|}])} \left(E_0(\rho_\mathrm{D}, \Pr([X_1,\ldots,X_{|\mathcal{X}|}] = [x_1,\ldots,x_{|\mathcal{X}|}]), 10\cdot\log_{10}(E_\mathrm{b}/N_0)) - \rho_\mathrm{D} R\right),$$

and the Gallager function

[6] This bound is also known as the *random coding bound* and the error exponent bound.

$$\mathrm{E}_0\left(\rho_\mathrm{D}, \Pr([X_1,\ldots,X_{|\mathcal{X}|}]=[x_1,\ldots,x_{|\mathcal{X}|}]), 10\cdot\log_{10}(E_\mathrm{b}/N_0)\right)$$
$$= -\mathrm{ld}\left[\int_{-\infty}^{\infty}\left(\sum_{i=1}^{|\mathcal{X}|}[\mathrm{PDF}_{Y|X}(y\mid x_i)]^{\frac{1}{1+\rho_\mathrm{D}}}\Pr(X_i=x_i)\right)^{1+\rho_\mathrm{D}}\mathrm{d}y\right],$$

follow the definitions known in literature [Gal68]. Note that $\mathrm{PDF}_{Y|X}(y \mid x_i)$, $i = 1,\ldots,|\mathcal{X}|$, depends on $10\cdot\log_{10}(E_\mathrm{b}/N_0)$. The Gallager bound is an existence bound stating that it is possible to transmit information using block codes of finite length n, given $\mathrm{E_r}(R) > 0$, The expected FER that has to be accepted reads $\mathrm{E}_\mathcal{C}\{\mathrm{FER}\} \leq \mathrm{e}^{(-n\cdot\mathrm{E_r}(R))}$. To use the random coding bound as a comparative value in Chapter 8, we determine the lowest SNR that allows to obtain the desired $\mathrm{E}_\mathcal{C}\{\mathrm{FER}\}$ from Equation (5.6) when n is specified. To this end, we have to optimize the parameters ρ_D and $10\cdot\log_{10}(E_\mathrm{b}/N_0)$ at the same time. This is done by iterative computer search [Pee98].

Further elaboration is needed if the Gallager bound shall be used to classify results on $\mathrm{E}_\mathcal{C}\{\mathrm{BER}\}$. As the considered bound provides values for the FER only, we calculate an equivalent $\mathrm{E}_\mathcal{C}\{\mathrm{FER}\}$, where we assume that codes in an ensemble produce $\hat{d}$ bit errors in an erroneous frame. Here $\hat{d}$ denotes an estimate on the minimum distance from rate and code length. In order to find $\hat{d}$, we use the Gilbert-Varshamov-bound [MS77, p. 34], which represents the average minimum distance within the ensemble of all possible codes of given rate and length. For binary codes, the Gilbert-Varshamov-Bound reads

$$\sum_{i=0}^{\hat{d}-2}\binom{n-1}{i} < 2^{n-k}. \tag{5.7}$$

For given n and rate R we find the largest $\hat{d}$ such that Eq. (5.7) is still satisfied. Using the estimated "average" minimum distance, we find $\mathrm{E}_\mathcal{C}\{\mathrm{BER}\} \approx \frac{\hat{d}}{n}\cdot\mathrm{E}_\mathcal{C}\{\mathrm{FER}\}$.

The sphere packing bound (SPB) [Sha59] differs substantially from the Gallager bound. It marks a strict lower bound on the FER and is thus a non-existence bound on the power efficiency, i.e. for a given rate and code length no code can underrun the performance values set by this bound. Again, we assume that a given FER needs to be met. Similar technical modifications as for the Gallager bound allow to present this bound for given BER values. This bound depends on the code length and thus represents a non-existence bound when n, R, and FER (or BER, respectively) are specified. The sphere packing bound is given by

$$\mathrm{FER} > \mathrm{FER}_\mathrm{SPB}(n,\theta,W),$$

with

$$\mathrm{FER}_{\mathrm{SPB}}(n,\theta,W) = \mathrm{Q}(\sqrt{n}W) + \frac{(n-1)\mathrm{e}^{-\frac{nA^2}{2}}}{\sqrt{2\pi}} \int_{\theta}^{\pi/2} (\sin(\phi))^{n-2} \mathrm{f}_n(\sqrt{n}W\cos\phi)\,\mathrm{d}\phi,$$

where $W = \sqrt{\frac{2E_s}{N_0}}$, and the family of functions $\mathrm{f}_n(x)$ reads

$$\mathrm{f}_n(x) = \frac{1}{2^{\frac{n-1}{2}}\Gamma(\frac{n+1}{2})} \int_0^{\infty} z^{n-1}\mathrm{e}^{-x'^2/2+x'x}\,\mathrm{d}x',\, x' \in \mathbb{R}, n \in \mathbb{N}.$$

In this context, $\Gamma(\cdot)$ denotes the Gamma function [Wei02]. Finding the input parameter θ which leads to the tightest bound is a tedious task. We choose an approximative solution by finding a θ^* that satisfies

$$\frac{\Gamma\left(\frac{n}{2}\right)(\sin(\theta^*))^{n-1}}{2\Gamma\left(\frac{n+1}{2}\right)\sqrt{\pi}\cos(\theta^*)}\left(1-\frac{\tan^2(\theta^*)}{n}\right) = 2^{-n\cdot R}.$$

For details on this bound we refer the reader to [Sha59]. To obtain the numerical results presented in this thesis we closely follow [WS07] and [Wie07].

5.2 Convolutional Codes

Convolutional codes [Eli55] are the second type of widely used channel codes. We introduce the convolutional codes used in this thesis along with appropriate encoders and decoders.

The prominent difference between convolutional codes and block codes is the fact that there exist no constraints on the length of a source and encoded word. We briefly review convolutional codes along with two optimal decoders, the *Viterbi* decoder [Vit67] and the *BCJR* decoder [BCJR74]. When using convolutional codes, all n encoded symbols are in general affected by the k source symbols. Convolutional codes do not obey a block structure but can be understood as a kind of "stream encoding". For convolutional codes of rate $R = 1/n$, only one source symbol is fed into the encoder at a time ($k = 1$). Unlike for block codes, the code gain originates from memory instead of long blocks. The process of encoding is conducted by a *finite state machine* [Mea55]. Starting in a given state, the encoder emits a first encoded symbol as soon as a source symbol is present. The encoder memorizes its input for a (possibly infinite) number of time steps and calculates subsequent encoded symbols as a function of the current and the stored source symbols. If a non-recursive encoder is used, this number is limited, depends on the memory of the code, and reads $\mathrm{ld}(|\mathcal{S}|)$ for a binary code, where $\mathcal{S}$ denotes the set of possible encoder states. Contrary, *recursive* encoders feed back the encoded signal to their memory. With this, source symbols have an influence an *all* subsequent encoded symbols, what resembles an infinite memory. For details on convolutional codes, the reader is referred to [LC04]. As the notation

in the literature of convolutional codes is not unique, we elaborate on the notation used in this thesis in the following. For simplicity we restrict our attention to codes of rate 1/2.

A convolutional code is specified by the integers $g_{0,\text{oct}}$ and $g_{1,\text{oct}}$, given in octal representation. To obtain an encoding transfer function, $g_{0,\text{oct}}$ and $g_{1,\text{oct}}$ have to be converted to a binary representation denoted by g_0 and g_1. As demonstrated for a convolutional code with 2^3 states in Example 5.2.1, each digit of the octal representation is converted separately to three digits in the binary representation.

Example 5.2.1

$$g_{0,\text{oct}} = 13 \quad \rightarrow \quad g_0 = 001\,011$$

$$g_{1,\text{oct}} = 17 \quad \rightarrow \quad g_1 = 001\,111$$

Each of the $\text{ld}(|\mathcal{S}|)$ binary symbols in the binary representation g_i, $i = 0, 1$ represents a coefficient of the D-transform[7] [GRS01] of the impulse response of a linear time-invariant (LTI) system $G_0(D) = g_{0,0} \oplus g_{0,1} \odot D \oplus g_{0,2} \odot D^2 \oplus \ldots \oplus g_{0,\text{ld}(|\mathcal{S}|)} \odot D^{\text{ld}(|\mathcal{S}|)}$. The leftmost bit, the most significant bit (MSB), represents the highest-order coefficient $g_{0,\text{ld}(|\mathcal{S}|)}$. Example 5.2.2 shows this step for the convolutional code with 2^3 states.

Example 5.2.2

$$g_0 = 001\,011 \quad \rightarrow \quad G_0(D) = \bigoplus_{i=0}^{p-1} g_{0,i} \odot D^i = D^3 \oplus D \oplus 1$$

$$g_1 = 001\,111 \quad \rightarrow \quad G_1(D) = \bigoplus_{i=0}^{p-1} g_{1,i} \odot D^i = D^3 \oplus D^2 \oplus D \oplus 1$$

For the simulations conducted in this work, the encoding is accomplished in a systematic, recursive manner, i.e. the source symbols leave the encoder unchanged together with parity bits that are calculated by the transfer function $G_0(D)/G_1(D)$. This leads directly to

$$X_{\text{par}}(D) = \frac{G_0(D)}{G_1(D)} \odot U(D), \tag{5.8}$$

what is equivalent to

$$G_1(D) \odot X_{\text{par}}(D) = G_0(D) \odot U(D), \tag{5.9}$$

[7] The D-transform is equivalent to the z-transform with $D := z^{-1}$.

and can be transformed to

$$g_{1,0} \odot X_{\text{par}}(D) = G_0(D) \odot U(D) \oplus \bigoplus_{i=1}^{p-1} g_{1,i} D^i \odot X_{\text{par}}(D). \tag{5.10}$$

With $g_{1,0} = 1$, Equation (5.10) can instantly be implemented as an LTI system in direct form III [GRS01]. This realization is presented in Figure 5.2.

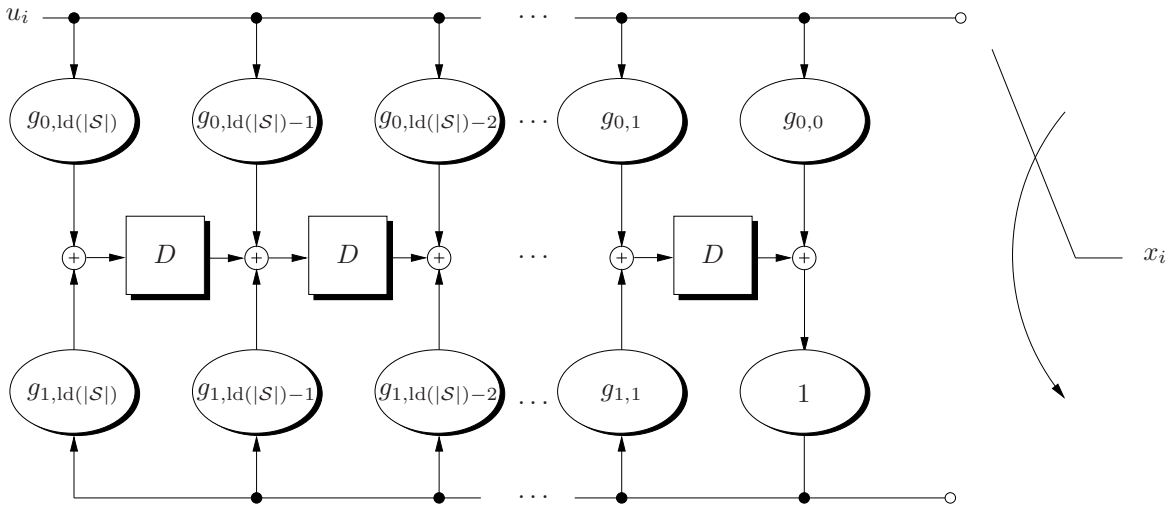


Figure 5.2: Systematic, recursive convolutional encoder with rate $R = 1/2$.

5.2.1 Convolutional Codes Used in this Thesis

In this thesis we use convolutional codes of rate 1/2 [Hue04, JZ99] with maximum free distance.

Definition 5.2.1 *[JZ99] The minimum Hamming distance between any two differing codewords of a convolutional code is called the free distance* d_{free}.

Table 5.4 lists the generator polynomials of the chosen codes along with the number of states $|\mathcal{S}|$ and the free distance. Codes with up to 2^{10} states are taken into account.

5.2.2 Decoding Convolutional Codes

The BCJR decoder [BCJR74] and the Viterbi decoder [Vit67] are both *optimal* approaches for decoding convolutional codes. The BCJR algorithm obtains optimal estimates for each transmitted source symbol (maximum-likelihood symbol-by-symbol estimation, MLSSE), whereas the output of the Viterbi decoder is the most probable *sequence* of estimated source symbols (maximum-likelihood sequence estimation, MLSE). We introduce both decoders only very briefly and assume that the reader is familiar with these decoders on the level of [LC04]. In particular, we emphasize a sliding-window approach of the BCJR decoder [Hub92] which allows to obtain good decoding results with low decoding delay. Later we transfer this approach to the Viterbi decoder.

$g_{0,\text{oct}}$	$g_{1,\text{oct}}$	Number of states $\lvert\mathcal{S}\rvert$	d_{free}
2	3	2^1	3
5	7	2^2	5
13	17	2^3	6
31	27	2^4	7
65	57	2^5	8
155	117	2^6	10
345	237	2^7	10
435	657	2^8	12
1671	1233	2^9	12
2731	2157	2^{10}	14

Table 5.4: Generator polynomials, the number of states $|\mathcal{S}|$ and the free distance d_{free} for the convolutional codes of rate 1/2 used in this thesis [Hue04, JZ99].

The BCJR decoder estimates state probabilities at time step i by forward and backward estimation and stores these values in $|\mathcal{S}|$-dimensional vectors $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$, respectively. In this context, we assume that k source symbols are transmitted. Consequently, the time index i needs to vary between $i = 0$, denoting the state probabilities before the first source symbol was observed, and $i = k$, specifying the state probabilities after the last source symbol. Note that for the backward recursion, the standard BCJR decoder must be informed about the state probabilities before the encoding of the sequence has started and after this task has been completed, i.e. $\boldsymbol{\alpha}_0$ and $\boldsymbol{\beta}_k$, and consequently about the complete received sequence from $i = 1$ to $i = k$. Usually $\boldsymbol{\alpha}_0$ and $\boldsymbol{\beta}_k$ are known to the receiver as the encoder starts and terminates encoding in the zero-state. The knowledge of the complete received sequence is required for decoding. This is required for decoding and hence the decoder cannot start before the whole sequence is received.

We describe next a variation of the BCJR decoder which works exclusively on a coherent sub-block of the received sequence. The sub-block is singled out by a sliding-window approach [Hub92], what allows to start the decoding process earlier. As a consequence, the estimation of the state probabilities and with this the whole decoding process becomes suboptimal. However, proper estimation of $\boldsymbol{\beta}$ allows for keeping the sub-optimality very low [HJ71]. There exists a trade-off between the imposed decoding delay and the decoding performance. In order to estimate u_i, it is sufficient to know y_1 to $y_{i+1+w_{\boldsymbol{\beta}}-1}$ which allows for estimation of $\boldsymbol{\beta}_i$. The accuracy of this estimation strongly depends on the number of time steps used for backward state estimation, $w_{\boldsymbol{\beta}}$. This variable is a user parameter. As all input sequences and consequently all decoder states are equally a-priori-probable, the entries in $\boldsymbol{\beta}_{\min(i+w_{\boldsymbol{\beta}},k)}$ are chosen to be equal what allows for estimation of $\boldsymbol{\beta}_i$ to $\boldsymbol{\beta}_{\min(i+w_{\boldsymbol{\beta}}-1,k)}$ by backward recursion. With this, it is possible to find $\hat{u}_i$. This method is known as the sliding-window BCJR [Hub92, p. 133]. A schematic visualization to identify the variables is given in Figure 5.3. In this context, $w_{\boldsymbol{\beta}} = 3$ is set and a convolutional code of rate 1/2 and 4 states is assumed. Denoted is u_i, the source symbol currently to estimate, as well as all received symbols taken into account. Bars indicate the values

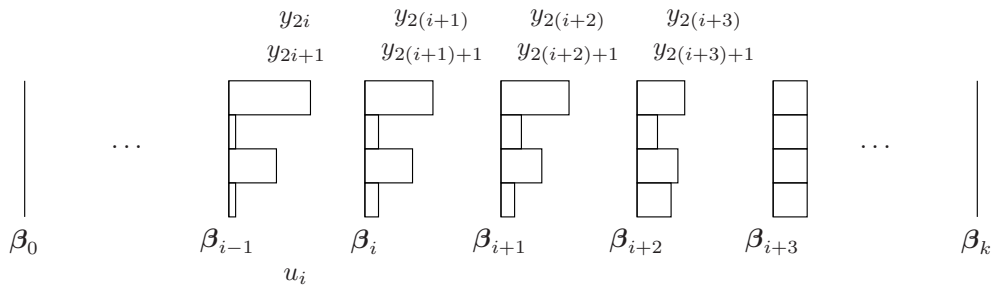


Figure 5.3: Sliding-window BCJR, $w_{\boldsymbol{\beta}} = 3$.

of the vectors $\boldsymbol{\beta}_i$ to $\boldsymbol{\beta}_{i+w_{\boldsymbol{\beta}}}$, where the latter one has a uniform distribution. The estimation of the vectors $\boldsymbol{\alpha}_i$, $i = 1, \ldots, k$, is accomplished in the standard manner [LC04].

Another option to decode a convolutional code is the Viterbi decoder [Vit67]. In this case, no backward recursion is required but this decoder also needs a fetch-ahead of received data for proper decoding. The reason for this is the fact that the decoder finds the optimal path through a trellis the following way. It uses an error metric to identify preferable states and marks paths in the trellis that allow to reach these states with lowest-possible error metric. Once this is done for the whole sequence, it selects the state with the lowest error metric in the last time step and estimates the transmitted sequence by following the marked paths backwards. Again, we introduce a sliding-window approach, which uses a subset of the received sequence only. Similar to $w_{\boldsymbol{\beta}}$ in the BCJR case, w_p denotes the number of time steps for which the decoder needs to have access to the corresponding received values in advance when processing the current received symbol. The decoder builds the path up to the most recent received symbol it has access to and decides on the source symbol currently to estimate. The increased reliability of the estimated source symbol is easy to see from a comparison between the direct decision ($w_\mathrm{p} = 0$) and $w_\mathrm{p} > 0$.

Example 5.2.3 *Let u_i be the source symbol of interest. In a direct decision at time step i, i.e. y_{2i} and y_{2i+1} are the most recent received symbols, the decision on the edge connecting the states at time step $i-1$ and i only depends on y_{2i} and y_{2i+1}. Contrary, with increasing look-ahead information, the choice on the state at time step i consolidates and with this, the edge between time step $i-1$ and i. The decision on this edge corresponds to the decision on the current source symbol.*

It was shown that the current estimate only has a satisfactory reliability if the user parameter w_p is not chosen too low [HJ71]. Both $w_{\boldsymbol{\beta}}$ and w_p are usually chosen as multiples of the constraint length for non-recursive encoders [JZ99], i.e. multiples of $\mathrm{ld}\,(|\mathcal{S}|) + 1$. Also for this decoder, we observe in subsequent chapters that the sliding-window approach obtains desirable performance results while keeping the advance access to received symbols limited.

5.2.3 Simulation Details for Convolutional Codes

We specify simulation details for convolutional codes such that the presented results can be reproduced easily. When measuring the BER and FER for convolutional codes, we only take source symbols into consideration. In order to measure the FER-performance for convolutional codes, one needs to specify the number of source symbols in a (virtual) frame. Usually, this number is given by a protocol running on a higher layer. We denote this value by L_{F}. Estimating the error gap distribution $g[\kappa]$, $\kappa = 1, \ldots, k$, with

$$\begin{aligned} g[\kappa] \;=\; & \Pr\Big((U_{k'} = u_{k'}) \neq (\hat{U}_{k'} = \hat{u}_{k'}), (U_{k'+1} = u_{k'+1}) = (\hat{U}_{k'+1} = \hat{u}_{k'+1}), \ldots \\ & \ldots (U_{k'+\kappa-1} = u_{k'+\kappa-1}) = (\hat{U}_{k'+\kappa-1} = \hat{u}_{k'+\kappa-1}), (U_{k'+\kappa} = u_{k'+\kappa}) \neq (\hat{U}_{k'+\kappa} = \hat{u}_{k'+\kappa}) \Big) \end{aligned}$$

by simulation allows to calculate the FER for different values of L_{F} with only one simulation run. This distribution specifies the estimated probabilities that there exist $\kappa - 1$ correctly estimated source symbols between two successive errors, what allows for calculation of the FER on a frame of length L_{F} by [Ado74, Hub82]

$$\mathrm{FER} = \mathrm{BER} \cdot \sum_{\kappa=1}^{L_{\mathrm{F}}} \left(1 - \sum_{\kappa'=1}^{\kappa-1} g[\kappa'] \right).$$

This tool is an enrichment when estimating the performance of a convolutional code over a large scope of values of L_{F}.

Chapter 6

Stopping Sets and Automorphism Decoding

In this chapter, we introduce *automorphism decoding*, a novel decoding scheme for the BEC. This approach allows for a decoding performance close to maximum-likelihood decoding when using codes of a rich algebraic structure. First, we discuss the structural entities which prevent the iterative decoder from successfully converging to a valid codeword. This leads us to the definition of the so called *stopping sets*. In literature, there exist solutions for avoiding the existence of stopping sets. All of them are based on providing the decoder with redundant parity information. This is usually done by using a redundant parity-check matrix. Contrary, we introduce a more elegant scheme, the automorphism decoding approach [HMLH07, HMLH08], which generates the redundant parity checks from the standard parity-check matrix on request. This method has a very low memory consumption and provides surprisingly good performance results.

6.1 Iterative Decoding on the BEC: the Edge-Removal Algorithm

Belief-propagation decoding after signaling over the BEC is a simple task, as the positions of all erased symbols are known. The typical decoding situation for the BEC is processing a received vector $\boldsymbol{y}$, which contains only error-free symbols or erasures. Example 3.1.1 and Example 3.2.1 discuss the variable-node operation and the check-node operation of a BP decoder for the special case of signaling over the BEC and show that both are straightforward operations. Actually, the information combining process recovers an erasure as soon as a non-erased symbol is available in form of extrinsic information, i.e. as soon as a non-erased symbol is passed to the according variable node. On the right-hand side of the Tanner graph, a check node is able to calculate extrinsic information for an erased symbol iff all other variables are not erased. Note that symbols recovered in this process are always correct and no contradictory information can be obtained from two adjacent edges as the BEC does not introduce errors.

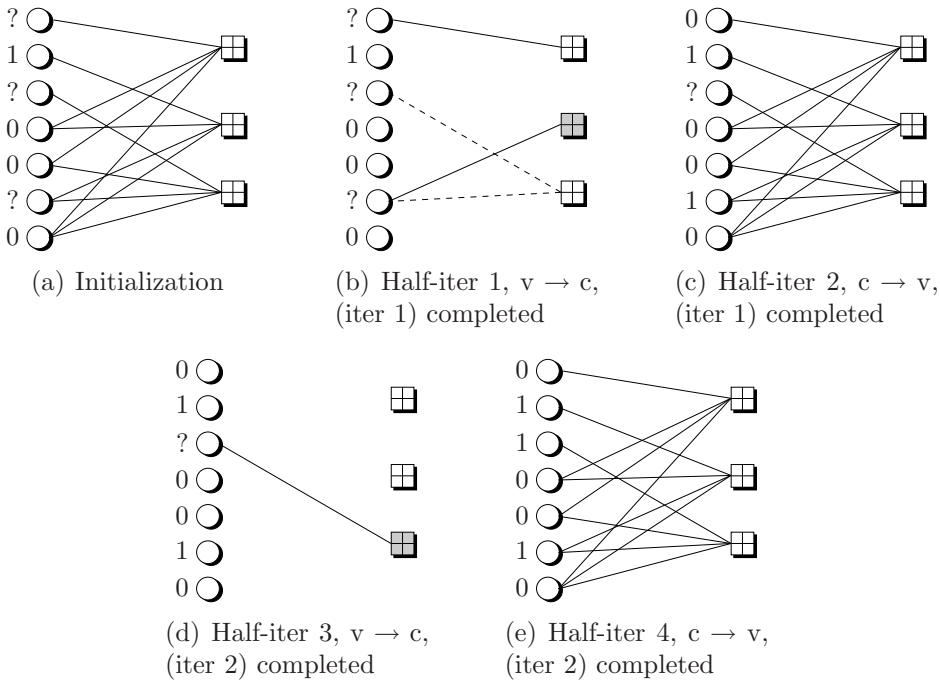


Figure 6.1: Successful BP decoding of a partially erased sequence

Example 6.1.1 *Figure 6.1 shows the decoding process using the parity-check matrix of the* $[7, 4, 3]$ *Hamming code.*

In a first half-iteration, all known information is sent from the variable nodes to the check nodes. These nodes calculate the modulo-2 sum of the corresponding received values, i.e.

$$\bigoplus_{j' \in \mathcal{N}(\mathrm{c}_j) \setminus \{i_?\}} x_{j'}, \tag{6.1}$$

where $\{i_?\}$ *denotes the set of erased positions. We use a special illustration feature in Figure 6.1. Whenever the passing of variable nodes to check nodes has been completed (Figures 6.1(b) and 6.1(d)), only the edges connected to erased variable nodes are depicted. We indicate the check nodes with* $\bigoplus_{j' \in \mathcal{N}(\mathrm{c}_j) \setminus \{i_?\}} x_{j'} = 1$ *by grey color and all others by white color (Figure 6.1(b)). Let us now describe the information transfer from the check nodes to the variable nodes. Only check nodes connected to exactly one erased variable node are able to report information to this variable node. To emphasize this entity, we indicate all edges connecting check nodes which cannot report information to the connected variable nodes by a dashed line. Figure 6.1(c) shows the decoding status after completion of this step. All variable nodes have replaced the erasures by the incoming information. All lines are now solid, as the variable nodes report to all check nodes in this step. This process repeats until all erasures are recovered or a maximum number of iterations is reached.*

In the preceeding algorithm, all edges removed in half-iteration 1 (Figure 6.1(b)) and half-iteration 3 (Figure 6.1(d)) were redrawn such that the data transfer in the subsequent variable-

to-check steps is properly visualized. The redrawing of the edges is required as the check nodes do not memorize the values calculated from previous half-iterations. If this kind of memory was available, the corresponding edges could be removed for good while the result of Equation (6.1) would only be updated by considering the positions in the neighborhood recovered in the previous half-iteration. For this reason, the algorithm described above is often referred to as the *edge-removal algorithm* [RU08].

6.2 Stopping Sets and Redundant Decoding

Assume that a BP decoder with a given matrix $\boldsymbol{H}$ is available for decoding a codeword transmitted over the BEC. It is straightforward to see that the decoder makes no more progress whenever a non-empty erasure pattern occurs that has at least two or no erasures connected to each check node. The number and size of possible erasure patterns with such a severe consequence for the iterative decoding process depend solely on the parity-check matrix. In the following, we define *stopping sets* [HMLH07, HMLH08] rigorously and state precisely what needs to happen on the channel such that the decoding process is affected by a set of this kind. Note that the notion of stopping sets was first presented in [DPT+02].

Definition 6.2.1 *Let $\{1, \ldots, n\}$ be the set of positions in a codeword and let $\boldsymbol{H}$ be a parity-check matrix of $\mathcal{C}$. A subset $\mathcal{P}_\sigma$, $|\mathcal{P}_\sigma| = \sigma$, of the set of positions in the codeword corresponds to a set of columns of $\boldsymbol{H}$. The set $\mathcal{P}_\sigma$ is a stopping set of size σ iff the restriction (cf. Definition 5.1.12) of $\boldsymbol{H}$ to the columns corresponding to $\mathcal{P}_\sigma$ does not contain rows of Hamming weight one.*

The sole existence of a stopping set does not necessarily affect the decoding process. A stopping set can be understood as a necessary but non-sufficient condition for the BP decoder to terminate the iterative decoding process without success.

Definition 6.2.2 *Given a parity-check matrix $\boldsymbol{H}$, we say that the BP decoder runs into a stopping set or encounters a stopping set, if all positions of a stopping set of $\boldsymbol{H}$ are erased and all other positions contain non-erased information.*

Note that in the above definition we assume all positions outside of the currently considered stopping set to be received correctly in order to assure that the performance of the decoder is impaired by exactly one stopping set.

As stated in Chapter 5, the parity-check matrix for a given code is not unique. For decoding a given received word, it can be desirable to prevent the decoder from running into one specific stopping set by directed altering of the parity-check matrix. If parity checks are replaced, these procedures may create new stopping sets at different coordinates. We will observe later that appending parity checks does not introduce new stopping sets.

Definition 6.2.3 *We say that the action modifying the current parity-check matrix resolves a stopping set if the corresponding columns of the parity-check matrix do not resemble a stopping set after this action has been accomplished. For convenience, we often say that the modification to the matrix itself (e.g. providing an additional row) resolves a stopping set.*

Assume for a moment that the parity-check matrix is not yet defined, i.e. that for the number of rows $m = 0$ is hypothetically assumed. In this case, all $\binom{n}{\sigma}$ possible stopping sets exist. We say that a parity-check matrix resolves a set of stopping sets to express a comparison between the set of all possible stopping sets and the set of stopping sets existing for the current matrix.

Example 6.2.1 *Consider the following parity-check matrix of the $[7, 4, 3]$ Hamming code (corresponding to Example 6.1.1), which is given by*

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

In the following, we discuss stopping sets of size 3. Focus on the restriction of $\boldsymbol{H}$ to the columns 1, 2, and 4, as well as the restriction to the columns 5, 6, and 7, which are shown below.

	$\boldsymbol{H}(:,1)$	$\boldsymbol{H}(:,2)$	$\boldsymbol{H}(:,4)$
$\boldsymbol{H}(1,:)$	1	0	1
$\boldsymbol{H}(2,:)$	0	1	1
$\boldsymbol{H}(3,:)$	0	0	0

	$\boldsymbol{H}(:,5)$	$\boldsymbol{H}(:,6)$	$\boldsymbol{H}(:,7)$
$\boldsymbol{H}(1,:)$	1	0	1
$\boldsymbol{H}(2,:)$	0	1	1
$\boldsymbol{H}(3,:)$	1	1	1

Recall from Definition 5.1.12 that we denote a restriction of a matrix $\boldsymbol{H}$ to the set $\mathcal{P}'$ as $\boldsymbol{H}(:, \mathcal{P}')$. Similarly, we denote the restriction of the transposed matrix to $\mathcal{P}'$ by $\boldsymbol{H}(\mathcal{P}', :)$. In the given example, we observe that both the restrictions to columns 1, 2, and 4, as well as the columns 5, 6, and 7 do not contain rows of weight 1. Hence, these restrictions correspond to stopping sets of size $\sigma = 3$ and our decoder encounters a stopping set if positions 1, 2, and 4 or positions 5, 6, and 7 are jointly erased within one received word. In total, the given parity-check matrix $\boldsymbol{H}$ contains 10 stopping sets of size $\sigma = 3$ and none of smaller size. Replacing parity check $\boldsymbol{H}(3, :)$ by the linear combination of $\boldsymbol{H}(2, :)$ and $\boldsymbol{H}(3, :)$ resolves the particular stopping set on column-position 5, 6, and 7.

Following the denomination in [WAG05], we define *dead-end sets* and also introduce ML-dead-end sets. Unlike stopping sets, dead-end sets do not immediately stop the progress of the decoder if the corresponding erasure pattern occurs, but prevent the decoder from recovering all erasures. The author appreciates the qualifier dead-end, as it reminds of a dead-end road which can be walked for a certain time but sooner or later prohibits one to advance further.

Definition 6.2.4 *A dead-end set is a subset of the set* $\{1, \ldots, n\}$ *which contains a stopping set. Define the support of a codeword as the set of its non-zero coordinates. An ML-dead-end set is a set which properly contains the support of a codeword* $\boldsymbol{x} \in \mathcal{C}$*.*

The preceeding definition considers the support of a codeword, what means that the all-zero codeword is used as a 'reference' codeword. The linearity of the block codes used in this thesis allows to restrict the attention to this special codeword as a reference codeword without loss of generality, what is demonstrated in the following example.

Example 6.2.2 *Given is a linear code of length* $n = 7$ *which contains the two codewords* $\boldsymbol{x}_1 =$ [1110101] *and* $\boldsymbol{x}_2 =$ [0011101]*. The ML decoder is able to choose the codeword by means of the received word unless positions* 1*,* 2*, and* 4 *are jointly erased. The linear combination of* $\boldsymbol{x}_1$ *and* $\boldsymbol{x}_2$ *shows that* [1101000] *is a valid codeword. Deciding between this codeword and the all-zero codeword is an equivalent problem to deciding between* $\boldsymbol{x}_1$ *and* $\boldsymbol{x}_2$*.*

If a codeword contains erasures on all positions corresponding to an ML-dead-end set, the ML decoder is not capable of recovering all these erasures. Hence, an ML-dead-end set is a dead-end set, but the converse claim is not true. We point out that dead-end sets are also often referred to as uncorrectable erasure patterns [HMLH08] or undecodable erasure patterns [HS07]. Dead-end set is more appropriate as the existence of such a set does not depend on the received erasure pattern but on the parity-check matrix. ML-dead-end sets are also known as ML-uncorrectable erasure patterns [HMLH08] or uncorrectable erasure patterns [HT07].

A valuable method to prohibit the decoder from running into stopping sets is to decrease their number by modifications of the parity-check matrix. More flexibility in the decoding process is obtained by including additional (redundant) parity checks into the decoding process. Such a parity check is supportive if it is connected to exactly one erased symbol and is able to adhere the iterative decoding process for at least one further iteration.

Adding a redundant parity check can only demote the local girth distribution, cf. Section 5.1.2 or leave it untouched – improving the local girth distribution is not possible. An example is given in the following. Figure 6.2 shows a section of a low-dense parity-check matrix, where only 1-entries are marked, and how appending the linear combination of lines 2 to 4 creates a bypass (dotted line) to the shown cycle of length 8 (dashed line). An impaired girth distribution usually leads to a worse decoding performance. However, in the case of BEC decoding, only correct information or erasures are fed back, the latter not affecting already decided symbols. Thus, additional feedback has no negative effect on the decoding performance. Contrary, appending additional rows helps avoiding stopping sets and is thus beneficial for the performance of the channel coding scheme. Many researchers proposed to use redundant parity-check matrices for the BEC in order to improve the decoding performance. Of course, strong interest exists in improving the performance significantly while augmenting only a small number of additional parity checks. This is based on the argument that supplemental parity checks require additional

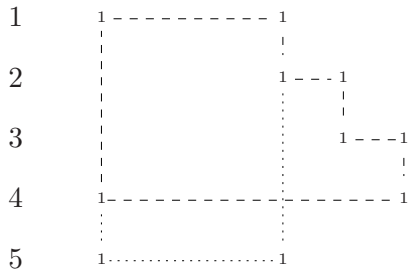


Figure 6.2: Cycle of length 8 (dashed line), shortened to a cycle of length 4 by an additional check (dotted line).

operations in each iteration and moreover an increased memory. Pioneering work on this subject was done by Schwartz and Vardy [SV06], who derived upper and lower bounds on the stopping redundancy. To define this properly, the stopping distance of a parity-check matrix needs to be introduced.

Definition 6.2.5 *The stopping distance of a parity-check matrix $\boldsymbol{H}$ is the cardinality of its smallest stopping set [SV06].*

An upper bound on the stopping distance is the minimum distance d of the underlying code. This is motivated by the fact that codewords correspond to ML-dead-end sets for which no further decoding progress is possible. These sets correspond to stopping sets which exist independently of the choice of the parity-check matrix. We point out that the stopping distance does not actually represent a distance measure. A distance is measured between two objects. In the case of the stopping distance, even the definition of these objects is missing. Nevertheless, this terminology is widely used in the coding theory literature[1]. The intention of this denomination is to show the parallel to the minimum distance for the AWGN case. The terminology "stopping distance" is adopted in this thesis.

In a next step, the stopping redundancy is defined.

Definition 6.2.6 *Let $\mathcal{C}$ be a linear code with minimum distance d. The stopping redundancy of $\mathcal{C}$ is the smallest non-negative integer $\rho_d(\mathcal{C})$ such that there exists a (possibly redundant) parity-check matrix $\boldsymbol{H}$ of $\mathcal{C}$ with $\rho_d(\mathcal{C})$ rows and stopping distance at least d [SV05, SV06].*

A straightforward lower bound on the stopping redundancy is given by the co-dimension of the code, $n - k$.

A constructive way to obtain upper bounds on the stopping redundancy is to find matrices with $\rho_d(\mathcal{C})$ rows and stopping distance d. Several authors presented results of that type as well as theoretical results. Both Schwartz and Vardy [SV06] as well as Han and Siegel [HS07] proposed redundant matrices for the $[24, 12, 8]$ Golay code of 34 rows with stopping distance 8 and derived

[1] Orlitsky et al. [OVZ03] introduced the term *stopping number* in 2003, Schwartz and Vardy [SV05] renamed it to *stopping distance* in 2005. Since then, stopping distance is common.

theoretical bounds. Also, Weber and Abdel-Ghaffar derived results in this context [WAG05], including an analytical description for the number of stopping sets in a parity-check matrix of a Hamming code. Hollmann and Tolhuizen [HT07] discussed the question of finding the redundancy to recover all erasure patterns which do not correspond to ML-dead-end sets up to a given size by means of iterative decoding. It is to observe that the stopping redundancy of the matrices constructed by the method introduced in [HT07] is comparably high and can be significantly underrun by the application of parity-check matrices of cyclic form, cf. Appendix C.2.3.

It is clear that especially small stopping sets impair the performance of a decoder significantly. For a stopping set of size σ to occur, the joint event of erasing all $\mathcal{P}_\sigma$ positions has to occur, thus the probability of a small stopping set is significantly higher, independent of the erasure probability. It may be sufficient to remove stopping sets of smaller size to obtain the decoding performance of interest. The stopping redundancy is not capable of identifying this and is therefore extended to the stopping redundancy hierarchy [HLMH06, HMLH07, HMLH08], defined below.

Definition 6.2.7 *Let $\mathcal{C}$ be a linear code with minimum distance d. For $\ell \leq d$, the ℓ-th stopping redundancy of $\mathcal{C}$ is the smallest non-negative integer $\rho_\ell(\mathcal{C})$ such that there exists a (possibly redundant) parity-check matrix $\boldsymbol{H}$ of $\mathcal{C}$ with $\rho_\ell(\mathcal{C})$ rows and stopping distance at least ℓ. The ordered set of integers*

$$[\rho_1(\mathcal{C}), \rho_2(\mathcal{C}), \rho_3(\mathcal{C}), \ldots, \rho_d(\mathcal{C})]$$

is called the stopping redundancy hierarchy of $\mathcal{C}$. The order of an element in the stopping redundancy hierarchy is its position in the list.

6.2.1 Lower Bounds on the Stopping Redundancy Hierarchy

We present next lower bounds on the ℓ-th stopping redundancy, $\ell = 4, \ldots, d$, of binary linear codes.

Theorem 6.2.8 *The third entry of the stopping redundancy hierarchy reads $\rho_3(\mathcal{C}) = n - k$ for any code with minimum distance at least 3.* [2]

Proof 6.2.9 *([SV06, Theorem 3]) Consider a binary linear code $\mathcal{C}$ that has minimum distance three or higher. As a consequence, any pair of two columns, taken from a full-rank parity-check matrix of such a code, is linearly independent. The restriction of the parity-check matrix to any pair of positions has at least one row of the pattern $0\,1$ or $1\,0$, or even both. Thus, $\rho_3(\mathcal{C}) = n - k$.* ■

In a next step, we present lower and upper bounds on the stopping redundancy hierarchy. In the following lemma, the operator $\mathrm{wt}(\cdot)$ denotes the weight of a vector.

[2] For codes with $d < 3$, the stopping redundancy $\rho_3(\mathcal{C})$ is not defined.

Lemma 6.2.10 *Let $\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)$ denote the set of stopping sets of size σ resolved by the i-th row of $\boldsymbol{H}$. The number of stopping sets of size σ, resolved by the i-th row of $\boldsymbol{H}$ with row-weight $\mathrm{wt}(\boldsymbol{H}(i,:))$, equals*

$$|\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| = \mathrm{wt}(\boldsymbol{H}(i,:)) \cdot \binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1}. \tag{6.2}$$

Proof 6.2.11 *Equation (6.2) is a consequence of the fact that a stopping set can be resolved a row which has weight one in the corresponding restriction. The number of possible stopping sets resolved by a row results from the product of $\mathrm{wt}(\boldsymbol{H}(i,:))$ choices for placing the one-entry and $\binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1}$ choices for placing the remaining zeros.* ∎

The first two lower bounds to be presented are generalizations and extensions of lower bounds on the stopping redundancy derived in [SV06]. We show the extension to $\rho_\ell(\mathcal{C})$ and point out that the results by Schwartz and Vardy follow for $\ell = d$.

Theorem 6.2.12 *Let $\mathcal{C}$ be a binary $[n,k,d]$ linear code, and let*

$$\theta_\sigma = \max\{\lceil (n+1)/\sigma \rceil - 1, d^\perp\},\ \sigma = 1,\ldots,d-1, \tag{6.3}$$

where $d^\perp$ denotes the minimum distance of the dual code $\mathcal{C}^\perp$. Then

$$\rho_\ell(\mathcal{C}) \geq \max\left(n-k, \max_{\sigma\in\{1,\ldots,\ell-1\}}\left(\frac{\binom{n}{\sigma}}{\theta_\sigma\binom{n-\theta_\sigma}{\sigma-1}}\right)\right). \tag{6.4}$$

Proof 6.2.13 *The proof closely follows the lines of the proof for Theorem 5 in [SV06], but applies to the stopping redundancy hierarchy and includes a more detailed explanation. For finding an upper bound on the stopping redundancy hierarchy, we compare the number of possible stopping sets and the number of stopping sets resolved by $\rho_\ell(\mathcal{C})$ rows. In order to estimate the latter number, we apply a union bound technique. According to Lemma 6.2.10, the number of stopping sets of size σ, resolved by a single parity check of weight $\mathrm{wt}(\boldsymbol{H}(i,:))$, reads*

$$\mathrm{wt}(\boldsymbol{H}(i,:))\binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1}.$$

Assuming that this holds for each single parity check of a matrix leads to an upper bound (union bound) on the total number of resolved stopping sets. An upper bound needs to exceed the possible number of stopping sets of size σ if $\rho_\ell(\mathcal{C})$ parity checks are used. This insight leads us to the equation

$$\binom{n}{\sigma} \leq \sum_{i=1}^{\rho_\ell(\mathcal{C})} \mathrm{wt}(\boldsymbol{H}(i,:))\binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1} \tag{6.5}$$

and with this to

$$\binom{n}{\sigma} \leq \rho_\ell(\mathcal{C}) \max_i \left(\mathrm{wt}(\boldsymbol{H}(i,:)) \binom{n - \mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma - 1} \right). \tag{6.6}$$

Note that in Equation (6.6) a maximization is required in order to find a lower bound on $\rho_\ell(\mathcal{C})$. As the weight of a parity-check equation is an integer number, we calculate the step-by-step slope [Wei02] of the function $\mathrm{wt}(\boldsymbol{H}(i,:)) \binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1}$ with respect to $\mathrm{wt}(\boldsymbol{H}(i,:))$ in order to find the maximum. The result reads

$$\begin{aligned} &\frac{(\mathrm{wt}(\boldsymbol{H}(i,:))+1)\binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))-1}{\sigma-1} - (\mathrm{wt}(\boldsymbol{H}(i,:)))\binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1}}{\mathrm{wt}(\boldsymbol{H}(i,:))+1-\mathrm{wt}(\boldsymbol{H}(i,:))} \\ &\qquad = \mathrm{wt}(\boldsymbol{H}(i,:)) \binom{n - \mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma - 1} \left(\frac{n - \sigma + 1 - \sigma \cdot \mathrm{wt}(\boldsymbol{H}(i,:))}{\mathrm{wt}(\boldsymbol{H}(i,:))(n - \mathrm{wt}(\boldsymbol{H}(i,:)))} \right). \end{aligned} \tag{6.7}$$

This allows us to observe that the function $\mathrm{wt}(\boldsymbol{H}(i,:)) \binom{n-\mathrm{wt}(\boldsymbol{H}(i,:))}{\sigma-1}$ rises with increasing parameter $\mathrm{wt}(\boldsymbol{H}(i,:))$, until it obtains its maximum at $\mathrm{wt}(\boldsymbol{H}(i,:)) = \theta_\sigma = \left\lceil \frac{n+1}{\sigma} \right\rceil - 1$, where the term in Equation (6.7) equals zero. The smallest possible weight of a parity check is $d^\perp$, what is considered by the maximum operation in Equation (6.3). Similarly, the maximum operation in Equation (6.4) assures that only matrices of full rank are deployed to find this bound. ■

The bound presented in Theorem 6.2.12 is not tight. Tighter lower bounds on the stopping redundancy hierarchy are presented next. To this end, we replace the union bound by more accurate bounding techniques, for example by invoking the principle of inclusion-exclusion [Doh03]. Although the principle of inclusion-exclusion is a very general concept, we restrict our attention to sets of events from a probability space.

Theorem 6.2.14 *Let $\mathcal{E} = \{E_1, E_2, \ldots, E_N\}$ be a set of events in an arbitrary probability space and let $\mathcal{B}$ be the set $\{1, 2, \ldots, |\mathcal{B}|\}$, where $|\mathcal{B}|$ denotes the cardinality of the set $\mathcal{B}$. For a set of sets $\{\mathcal{E}_b\}_{b\in\mathcal{B}}$, the principle of inclusion-exclusion (PIE) asserts that*

$$\left| \bigcup_{b\in\mathcal{B}} \mathcal{E}_b \right| = \sum_{\mathcal{I}\subseteq\mathcal{B}, \mathcal{I}\neq\emptyset} (-1)^{|\mathcal{I}|-1} \left| \bigcap_{b\in\mathcal{I}} \mathcal{E}_b \right|. \tag{6.8}$$

In this context, $\mathcal{I}$ is used to denote a subset, and $\emptyset$ is the empty set. A proof for Theorem (6.2.14) and more details on the PIE can be found in [Doh03].

Let $\mathrm{ind}(\underline{\mathcal{C}}) = \{1, 2, \ldots, m\}$ denote the set of row indices of $\boldsymbol{H}$. We introduce $\mathrm{sub}_j(\mathrm{ind}(\underline{\mathcal{C}}))$ to identify the set of all subsets with j elements of $\mathrm{ind}(\underline{\mathcal{C}})$. The number of stopping sets of size σ resolved by $\boldsymbol{H}$ can be found from Equation (6.8) as

$$\left| \bigcup_{i=1}^{m} \Sigma_\sigma\left(\boldsymbol{H}(i,:)\right) \right| = \sum_{\mathcal{I}\subseteq\mathrm{ind}(\underline{\mathcal{C}}), \mathcal{I}\neq\emptyset} (-1)^{|\mathcal{I}|-1} \left| \bigcap_{i\in\mathcal{I}} \Sigma_\sigma\left(\boldsymbol{H}(i,:)\right) \right| = \sum_{j=1}^{m} (-1)^{j-1} S_{\sigma,j}, \tag{6.9}$$

where $S_{\sigma,j} = \sum_{\mathcal{E} \in \mathrm{sub}_j(\mathrm{ind}(\underline{\mathcal{C}}))} s_{\sigma,\mathcal{E}}$. Here, $s_{\sigma,\mathcal{E}}$ denotes the number of stopping sets of size σ resolved by the rows indexed by elements of the set $\mathcal{E}$.

Let us now determine the ℓ-th stopping redundancy hierarchy of a code by means of the number of stopping sets resolved by the rows of the parity-check matrix. Again, we identify the smallest number m such that the cardinality of the union of stopping sets resolved by the single rows equals the number of possible stopping sets. Note that this has to be satisfied for all stopping set sizes taken into account by the currently considered element of the stopping redundancy hierarchy. In other words, we need to find m such that

$$|\bigcup_{i=1}^{m} \Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| = \binom{n}{\sigma}, \;\; 1 \leq \sigma < \ell. \tag{6.10}$$

Calculating the exact number of stopping sets based on the principle of inclusion-exclusion is a tedious task. Special straightforward cases exist, e.g. when all collections of j codewords from $\mathcal{C}^\perp$ resolve the same number of stopping sets. For this reason, the PIE is often used as an upper or lower bound by neglecting terms of higher order, i.e. terms for which $|\mathcal{I}| \geq b$, $b \in \mathbb{N}$ [Doh03]. As it can be shown that the absolute values of these terms decrease with increasing b, two kinds of bounds are possible. An upper bound is obtained if the sign of the first omitted term is negative. Otherwise, one obtains a lower bound. We refer to this type of inequalities as *Bonferroni* inequalities [Doh03, Kwe75].

We make use of the rich set of sophisticated Bonferroni inequalities from [Doh03], and we use an upper bound with $b = 2$ for stopping set analysis. This type of Bonferroni inequality reads

$$\left|\bigcup_{b\in\mathcal{B}} \mathcal{E}_b\right| \leq \sum_{\mathcal{I}\subseteq\mathcal{B},\,|\mathcal{I}|=1} (-1)^{|\mathcal{I}|-1} \left|\bigcap_{b\in\mathcal{I}} \mathcal{E}_b\right| + \frac{2}{|\mathcal{B}|} \cdot \sum_{\mathcal{I}\subseteq\mathcal{B},\,|\mathcal{I}|=2} (-1)^{|\mathcal{I}|-1} \left|\bigcap_{b\in\mathcal{I}} \mathcal{E}_b\right|. \tag{6.11}$$

Note that the preceeding equation is an upper bound, whereas a straightforward specialization of Equation (6.8) to $|\mathcal{I}| \leq 2$ would have lead to a lower bound. Let us now specialize the event sets $\mathcal{E}$ to sets of stopping sets.

Proposition 6.2.15 *A parity-check matrix with m rows resolves*

$$|\bigcup_{i=1}^{m} \Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| \leq \sum_{i=1}^{m} |\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| - \frac{2}{m} \cdot \sum_{i,j=1,\,i<j}^{m} |\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right) \cap \Sigma_\sigma\left(\boldsymbol{H}(j,:)\right)| \tag{6.12}$$

stopping sets of size σ. Equation (6.12) is an upper bound on the number of resolved stopping sets of size σ and a function of m. The lower bound on the stopping redundancy hierarchy follows from finding the smallest m such that

$$\binom{n}{\sigma} \leq \sum_{i=1}^{m} |\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| - \frac{2}{m} \cdot \sum_{i,j=1,\, i<j}^{m} |\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right) \cap \Sigma_\sigma\left(\boldsymbol{H}(j,:)\right)| \;\forall \sigma < \ell. \tag{6.13}$$

This inequality follows from the fact that the upper bound on the union of stopping sets resolved by the rows of the parity-check matrix needs to exceed the total number of possible stopping sets. Consequently, $\rho_\ell(\mathcal{C}) \geq m$.

In order to evaluate the lower bound, we now restrict our attention to the intersection numbers in Equation (6.13).

Definition 6.2.16 *Let $\mathcal{O}_l$ and $\mathcal{Z}_l$, $l \in \{1, \ldots, m\}$, denote the sets of positions of ones and zeros in the l-th row of the parity-check matrix $\boldsymbol{H}$ with m rows, respectively. The intersection number $|\mathcal{Q}_i \cap \mathcal{Q}'_j|$ of $\boldsymbol{H}$, $i, j \in \{1, \ldots, m\}$, is defined as the cardinality of the set $\mathcal{P}' \subseteq \{1, \ldots, n\}$ of positions p such that $H(i,p) = Q$ as well as $H(j,p) = Q'$ holds for all $p \in \mathcal{P}'$. In this context, Q and Q' are placeholders for O, marking a one entry, and Z, which stands for a zero, respectively. Similarly, $\mathcal{Q}$ and $\mathcal{Q}'$ are placeholders for the sets $\mathcal{O}$ and $\mathcal{Z}$.*

Lemma 6.2.17 *The number of stopping sets of size σ, resolved jointly by two rows of $\boldsymbol{H}$ indexed by i and j, equals*

$$|\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right) \cap \Sigma_\sigma\left(\boldsymbol{H}(j,:)\right)| = |\mathcal{O}_i \cap \mathcal{O}_j| \cdot \binom{|\mathcal{Z}_i \cap \mathcal{Z}_j|}{\sigma - 1} + |\mathcal{O}_i \cap \mathcal{Z}_j| \cdot |\mathcal{Z}_i \cap \mathcal{O}_j| \cdot \binom{|\mathcal{Z}_i \cap \mathcal{Z}_j|}{\sigma - 2}. \tag{6.14}$$

Proof 6.2.18 *The result in Equation (6.14) follows from the fact that there exist two possibilities how two non-equal rows can jointly remove a set of stopping sets. First, both rows share a "1" and all the "0"s in the remaining positions of the stopping set. As a second option, the two rows have common "0" symbols in all but two positions, while the remaining positions show the two alternating patterns containing a "0" symbol and a "1" symbol.* ∎

The expressions in Lemma 6.2.10 and Lemma 6.2.17 can be simplified significantly by assuming that all rows in $\boldsymbol{H}$ have weight ω (provided that $\mathcal{C}^\perp$ has sufficiently many codewords of weight ω), and by introducing the variable

$$|\Sigma_P| = \max_{\{i,j\}} |\Sigma_\sigma\left(\boldsymbol{H}(i,:)\right) \cap \Sigma_\sigma\left(\boldsymbol{H}(j,:)\right)|. \tag{6.15}$$

Under these assumptions, Proposition 6.2.15 can be rewritten as

$$|\bigcup_{i=1}^{m} \Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| \leq m \cdot \omega \cdot \binom{n-\omega}{\sigma-1} - \frac{2}{m} \cdot \binom{m}{2} \cdot |\Sigma_P|, \tag{6.16}$$

where $|\Sigma_P|$ can be found according to Equation (6.14) and Equation (6.15). Assuming constant row-weights, we find a lower bound on the ℓ-th stopping redundancy using Lemma 6.2.17 and Proposition 6.2.15. Again, this derivation is based on the principle of observing that the upper bound in (6.16) has to exceed $\binom{n}{\sigma}$ for all $\sigma \leq \ell$. As a consequence,

$$\rho_\ell(\mathcal{C})\big|_{\mathrm{wt}(\boldsymbol{H}(i,:))=\omega} \geq \max_{\sigma<\ell} \left\lceil \frac{\binom{n}{\sigma} - |\Sigma_P|}{\omega \binom{n-\omega}{\sigma-1} - |\Sigma_P|} \right\rceil , \tag{6.17}$$

where the qualifier $\mathrm{wt}(\boldsymbol{H}(i,:)) = \omega$ emphasizes that a constant row weight ω in $\boldsymbol{H}$ was assumed for this bound.

We deduct from the previous derivations that lowering the number of stopping sets which are resolved jointly by several subsets of rows is a good strategy to obtain a desirable stopping redundancy hierarchy. Therefore, small sizes for the intersections of the supports of pairs, triples, etc., of rows are desirable. This observation supports the fact that parity-check matrices of low weight are most appropriate for BP decoding. We henceforth restrict our attention to matrices $\boldsymbol{H}$ that contain minimum weight codewords of $\mathcal{C}^\perp$ as their rows.

Example 6.2.3 *Given is a parity-check matrix of the form shown below,*

$$\boldsymbol{H} = \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 1 \end{array}\right] .$$

Let us show that this is a suboptimal choice of a parity-check matrix as several stopping sets are removed by more than one row. For this purpose, we consider row 2 *and row* 3 *in the preceeding matrix. The columns indexed by* $\{1, 4, 5\}$ *have the pattern "100" as a common restriction for both rows. The columns indexed by* $\{2, 3, 4\}$ *and* $\{2, 3, 5\}$ *have row restrictions "010" and "100", respectively. Consequently, both rows simultaneously resolve stopping sets indexed by the described set of columns. Subsuming, both rows resolve the three restrictions indexed by* $\{1, 4, 5\}$*,* $\{2, 3, 4\}$*, and* $\{2, 3, 5\}$*, while all other restrictions are resolved by at most one of these two rows.*

6.2.2 Upper Bounds on the Stopping Redundancy Hierarchy

We derive upper bounds on the stopping redundancy hierarchy using probabilistic methods and generalizations of the bounds presented in [SV06].

Before presenting the bounds, let us briefly introduce Lovász Local Lemma (LLL) [EL75, AS00], as well as its high-probability variation.

Lemma 6.2.19 *Let $\mathcal{E}$ denote a set of events in an arbitrary probability space. Suppose that each event E_i is independent of all other events $E_{i'}$, except for at most j of them, and that*

$$P\{E_i\} \leq p, \ \ \forall\, 1 \leq i \leq |\mathcal{E}|\,. \tag{6.18}$$

If

$$\mathrm{e}\, p\, (j+1) \leq 1, \tag{6.19}$$

then $P\{\bigcap_{i=1}^{|\mathcal{E}|} \overline{E}_i\} > 0$, where $\overline{E}_i$ is the counter event to E_i. Based on the local events listed in Equation (6.18) and the additional constraint in Equation (6.19), the Lovász Local Lemma makes a statement on a more "global" event. In particular, it says that for a set of events, each with limited occurrence probability and not all of them being dependent, it is possible that none of the events happens.

Let us now introduce the high-probability variation of Lovász Local Lemma. Let $0 < \epsilon < 1$. If

$$P\{E_i\} \leq \frac{\epsilon}{|\mathcal{E}|}\left(1 - \frac{\epsilon}{|\mathcal{E}|}\right)^{j}, 1 \leq i \leq |\mathcal{E}|, \tag{6.20}$$

then $P\{\bigcap_{i=1}^{|\mathcal{E}|} \overline{E}_i\} > 1 - \epsilon$. Loosely speaking, this variation asserts that under certain conditions, it is very probable that none of the events happens.

Based on Lovász Local Lemma and its high-probability variation (Lemma 6.2.19), one can obtain the following upper bounds on the ℓ-th stopping redundancy of a code. Note that the restriction $\ell \leq \lfloor \frac{d+1}{2} \rfloor$ holds.

Theorem 6.2.20 *Let $\mathcal{C}$ be an $[n, k, d]$ code. If $\ell \leq \lfloor \frac{d+1}{2} \rfloor$ and*

$$m \geq \frac{1 + \log \sum_{j=1}^{\ell-1} \left(\binom{n}{j} - \binom{n-j}{j} \right)}{-\log\left(1 - \frac{\ell-1}{2^{\ell-1}}\right)} + n - k - \ell + 1, \tag{6.21}$$

then $\rho_\ell(\mathcal{C}) \leq m$. Alternatively, if the conditions of the theorem are fulfilled, then there exists at least one parity-check matrix with m rows and stopping distance at least ℓ.

The proof of this non-constructive bound as well as an asymptotic estimate of the above bound for codes with $d = \text{const.} \cdot n$ is given in [HMLH08]. The idea behind Theorem 6.2.20 is to identify the events E_i with the events that a certain set of columns in a randomly drawn matrix corresponds to a stopping set. Once this identification is accomplished, inequality $P\{\bigcap_{i=1}^{|\mathcal{E}|} \overline{E}_i\} > 1 - \epsilon$ can be used to make a statement on the stopping redundancy hierarchy.

One way to make the finding of Theorem 6.2.20 more useful for practical purposes is to generalize it by invoking the high probability version of LLL, as stated below.

Theorem 6.2.21 *Let $\mathcal{C}$ be an $[n, k, d]$ code, let $\ell \leq \lfloor \frac{d+1}{2} \rfloor$, and assume that m fulfills the condition*

$$m \geq \frac{\log \frac{\epsilon}{\sum_{j=1}^{\ell-1} \binom{n}{j}} + \left(\sum_{j=1}^{\ell-1} \left[\binom{n}{j} - \binom{n-j}{j} - 1 \right] \right) \cdot \log \left(1 - \frac{\epsilon}{\sum_{j=1}^{\ell-1} \binom{n}{j}} \right)}{\log \left(1 - \frac{\ell-1}{2^{\ell-1}} \right)}. \tag{6.22}$$

If Equation (6.22) is fulfilled, the probability that a parity-check matrix consisting of m randomly chosen codewords of the dual code (with possible repetitions of a codeword) has stopping distance ℓ is at least $1-\epsilon$. With at most $n-k-\ell+1$ further rows appended to the m existing rows, the matrix also has rank $n-k$ and represents a valid parity-check matrix of the code $\mathcal{C}$.

The proof of the theorem is again given in [HMLH08]. There are two drawbacks of the results of Theorems 6.2.20 and 6.2.21. First, they are non-constructive and second, they are conditioned on $\ell \leq \lfloor \frac{d+1}{2} \rfloor$. For these reasons we present another method for finding upper bounds on the ℓ-th stopping redundancy. This bound was originally found in [SV06] and is extended in this thesis.

Theorem 6.2.22 *([SV06]) Let $\mathcal{C}$ be a binary linear code with parameters $[n,k,d]$, with $d>3$. Then*

$$\rho_d(\mathcal{C}) \leq \binom{n-k}{1} + \ldots + \binom{n-k}{d-2}. \tag{6.23}$$

In a next step, we generalize this result to the stopping redundancy hierarchy.

Theorem 6.2.23 *Let $\mathcal{C}$ be a binary linear code with parameters $[n,k,d]$, with $d>3$. Then*

$$\rho_\ell(\mathcal{C}) \leq \binom{n-k}{1} + \ldots + \binom{n-k}{\ell-2}, \ \ell \leq d. \tag{6.24}$$

The idea behind Equation (6.24) is based on a simple code property. Recall that in a code $\mathcal{C}$ with minimum distance d, each choice of $1 \leq i \leq d-1$ non-equal columns of a parity-check matrix represents a set of linearly independent column vectors. Consequently, any restriction of the matrix to $1 \leq i \leq d-1$ positions has rank i. Let us now construct a matrix which contains all possible patterns within each restriction to $1 \leq i \leq d-1$ positions. This requires to generate all linear combinations of up to $1 \leq i \leq d-1$ rows which are chosen from the existing $n-k$ rows. In this context, the knowledge on the rank of the matrix allows us to stop at $i=d-1$. Straightforward, the generated matrix includes all vectors which have weight 1 within a restriction to $i \leq d-1$ positions. As only one vector of weight one for each restriction is required, a little technical modification described in [SV06] allows to stop the process of generating linear combinations already at $i=d-2$. This leads to the right-hand side of Equation (6.23). The same argument leads to Equation (6.24). In this context, d needs to be replaced by ℓ.

In the following, we use subcodes of linear block codes to find tight upper bounds for single elements of the stopping redundancy hierarchy. Note that a linear block code is generally understood as a subspace of the vector space $\mathbb{F}_2^n$. Further, a subcode of a linear block code is a subspace of this code and hence again a linear block code. In the following, we make use of the

fact that the generators of $\mathcal{C}_1^\perp$ form a subset of the generators of $\mathcal{C}^\perp$, where we assume that $\mathcal{C}_1^\perp$ is a subcode of $\mathcal{C}^\perp$. This leads to the following interesting result.

Theorem 6.2.24 *Let $\mathcal{Y}$ be the set of all subcodes of the dual code $\mathcal{C}^\perp$ of a linear $[n,k,d]$ code $\mathcal{C}$ that have support weight[3] n and dual distance (cf. Definition 5.1.2) ℓ. Furthermore, let the dimensions of the subcodes in $\mathcal{Y}$ be $k_i, i = 1,\ldots,|\mathcal{Y}|$, and define $k_{\min} = \min_i k_i$. Then it is sufficient to apply Equation (6.23) to the subcode with the lowest dimension to obtain stopping distance ℓ. Consequently,*

$$\rho_\ell(\mathcal{C}) \leq \binom{k_{\min}}{1} + \ldots + \binom{k_{\min}}{\ell-2}. \tag{6.25}$$

Proof 6.2.25 *Let $\mathcal{C}_1$ be an $[n, n-k_1, \ell]$ code, and let its dual code $\mathcal{C}_1^\perp$ be a subcode of $\mathcal{C}^\perp$ of support weight n and dimension k_1. From Equation (6.24) we have $\rho_\ell(\mathcal{C}_1) \leq \binom{k_1}{1} + \ldots + \binom{k_1}{\ell-2}$. As $\mathcal{C}_1^\perp$ is a subcode of $\mathcal{C}^\perp$, the set of rows of the parity-check matrix of $\mathcal{C}_1$ forms a subset of the rows of the parity-check matrix of $\mathcal{C}$. As a result, the upper bound also holds for $\rho_\ell(\mathcal{C})$. This holds for any $\mathcal{C}_i$ and k_i, $i = 1,\ldots,|\mathcal{Y}|$. As the function $\binom{k_i}{j}$ strictly increases with rising k_i, the tightest upper bound of this form is obtained by choosing $k_i = k_{\min}$.* ∎

Example 6.2.4 *Let $\mathcal{C}_{\text{Golay}}$ be the $[24,12,8]$ extended Golay code. The code $\mathcal{C}_{\text{Golay}}$ is self-dual and contains a $[24,10,8]$ subcode. This code is unique [Jaf], and has minimum dual distance 6. Therefore*

$$\rho_6(\mathcal{C}_{\text{Golay}}) \leq \binom{10}{1} + \binom{10}{2} + \binom{10}{3} + \binom{10}{4} = 385. \tag{6.26}$$

The upper bound in Equation (6.26) is a significant improvement compared to the preceeding results. Direct evaluation of Equation (6.23) leads to $\rho_6(\mathcal{C}) \leq \rho_8(\mathcal{C}) = 2509$. Evaluating Equation (6.24) results to $\rho_6(\mathcal{C}) \leq 793$. However, bounds found by the construction of parity-check matrices are still superior. The best known upper bound (found through extensive computer search, [SV06]) reads $\rho_8(\mathcal{C}_{\text{Golay}}) \leq 34$.

A tighter upper bound on the stopping redundancy hierarchy is obtained whenever one finds a representation which has a better stopping redundancy. In Appendix C we observe that parity-check matrices of cyclic form lead to a good stopping redundancy hierarchy if cyclic codes are used. We focus on this class of codes in the remainder of this chapter, as these codes can be decoded well with the automorphism decoding approach, to be introduced in the next section.

6.3 Permutation Decoding and Automorphism Decoding

In the remainder of this chapter we discuss a different kind of redundant decoding. Contrary to providing a redundant parity-check matrix, the decoder is equipped with information on the

[3]The support weight of a subcode of a code is defined as the number of positions for which at least one of the codewords of the subcode is non-zero.

structure of the code. This allows the decoder to generate a set of "virtual" redundant parity-check equations or to perform operations on the received word. This is done on request, i.e. when the decoder notices that a stopping set has occurred while decoding with the current parity-check matrix. Such an event is easily detectable from the decoding process by observing the number of erasures connected to each check node. To understand the given approach in detail, we first discuss permutations and the automorphism group of a code. Further, we review a decoding approach for the binary symmetric channel (BSC) which is known from literature [MS77] and uses permutations to improve the decoding performance.

Definition 6.3.1 *([MS77, Ch. 8]) The set of permutations which send a code $\mathcal{C}$ into itself, i.e. permutations that map codewords into (possibly different) codewords, is called the automorphism group of the code $\mathcal{C}$, and is denoted by* $\mathrm{Aut}(\mathcal{C})$.

We point out that the automorphism group of a code contains permutations, whereas a general automorphism group contains functions [Wei02]. We are interested in a set of permutations, each element of which sends the *whole code* $\mathcal{C}$ into itself. This is clearly to distinguish from automorphisms on single codewords. These kinds of mappings would send specific codewords into itself and would be no use for a decoder which needs to be able to handle any codeword as a received word. Due to the restriction to linear codes $\mathcal{C}$, the automorphism group of a code and its dual code are equal, i.e.

$$\mathrm{Aut}(\mathcal{C}) = \mathrm{Aut}(\mathcal{C}^{\perp}) \tag{6.27}$$

holds [MS77, Ch. 8]. As a consequence, all valid permutations of the automorphism group of the code may also be applied to codewords of the dual code. Keep in mind that this set includes all valid parity-check equations of the code.

Let us now show that the discussed permutations are indeed automorphisms on the code and that they form a group. Before this is accomplished, we define an automorphism in a more general way.

Definition 6.3.2 *([Wei02]) An automorphism is an isomorphism of a system of objects onto itself.*

The isomorphism, already touched in Remark 3.1.3, is a very general mathematical concept and is formally defined as a *bijective morphism*. That means that both the mapping and its inverse are structure-preserving. In the following, we present an example.

Example 6.3.1 *Let $\mathbb{R}^+$ denote the positive-valued real numbers. The most common example for an isomorphism is the* log *function which is a structure-preserving mapping between the group $(\mathbb{R}^+, \cdot)$ and the group $(\mathbb{R}, +)$. This can easily be verified as the logarithmic function is a bijective mapping between $\mathbb{R}^+$ and $\mathbb{R}$. Furthermore, $\log(i \cdot j) = \log(i) + \log(j)\ \forall i, j \in \mathbb{R}^+$ holds.*

Note that isomorphisms do not only exist between groups but between objects in general. In our particular case using channel codes, only one object, the code $\mathcal{C}$ exists. We identify the isomorphism as the permutation operation which is applied to $\mathcal{C}$, and hence to each of its codewords, resulting in $\mathcal{C}$ again. As all permutations are invertible, the mapping is a bijection for sure and hence the considered operation is an automorphism. It still needs to be shown that the set of automorphisms for a code fulfills the requirements of a group [Wei02, p. 1267], what is accomplished in Appendix B. In this context, we define the concatenation of permutations as the group operation. It is to observe that the automorphism group is even an Abelian group.

6.3.1 Permutation Decoding

Decoding procedures using permutations from the automorphism group of a code in order to improve the decoding performance are known in literature [MS77, p. 513]. Procedures of this kind are referred to as *permutation decoding algorithms*. These schemes were invented for and applied to decoding of messages signaled over the BSC. Let us for a moment assume that we are transmitting over a BSC and that a codeword $\boldsymbol{x} \in \mathcal{C}$ is corrupted by an error vector $\boldsymbol{e} = [e_1,\ e_2,\ \ldots,\ e_n]$ of weight less than or equal to t_d, i.e. $\boldsymbol{y} = \boldsymbol{x} \oplus \boldsymbol{e}$, $\mathrm{wt}(\boldsymbol{e}) \leq t_d$. Here, t_d denotes the number of errors correctable by bounded minimum distance decoding (BMD), where $2t_d + 1 \leq d$ holds. It is the aim of the permutation decoding scheme to transform the received word using a permutation from $\mathrm{Aut}(\mathcal{C})$ such that erroneous symbols are only present in the positions carrying parity symbols. Recall that we use only binary codes in this thesis, cf. page 35. Consequently, both the generator matrix and parity-check matrix of any code considered here are from a Galois field of order and characteristic two[4]. We choose the parity-check matrix and the generator matrix of the special form

$$\boldsymbol{H} = [\boldsymbol{P} \mid \boldsymbol{I}_{n-k}], \quad \boldsymbol{G} = [\boldsymbol{I}_k \mid \boldsymbol{P}^{\mathsf{T}}],$$

where $\boldsymbol{I}_i$ is the identity matrix of size $i \times i$. This choice allows to deduct interesting conclusions on the error pattern by means of the syndrome. Note that the special choice of the generator matrix realizes a systematic encoding with $\mathcal{P}_{\mathrm{syst}} = \{1, \ldots, k\}$, $\mathcal{P}_{\mathrm{par}} = \{k+1, \ldots, n\}$. We refer to the positions in $\mathcal{P}_{\mathrm{par}}$ as parity positions, and $\mathcal{P}_{\mathrm{syst}}$ is defined in Section 5.1.3.

In general, the syndrome $\boldsymbol{s}$ reports symptoms of an erroneous transmission. It equals the zero vector of length $n - k$ if a valid codeword is received and otherwise returns the binary column sum of the restriction of $\boldsymbol{H}$ to the positions where $\boldsymbol{e}$ contains non-zero elements,

$$\boldsymbol{s} = \boldsymbol{y} \odot \boldsymbol{H}^{\mathsf{T}}.$$

[4] A Galois field of characteristic two has an order of $q = 2^a$, $a \in \mathbb{N}$

Obviously, the information provided by the syndrome is in general too limited to identify the error vector. However, due to the special choice of the parity-check matrix, it can be shown that the weight of the syndrome reveals information about the position of the errors.

Theorem 6.3.3 *Assume that the received vector $\boldsymbol{y}$ contains at most t_d errors. If the weight of $\boldsymbol{s}$ is greater than t_d, at least one error is located within the first k positions of $\boldsymbol{y}$. If the weight is less than or equal to t_d, all errors, if existing, are located in the parity positions $k+1$ to n.*

Proof 6.3.4 *([MS77]) Split the error vector $\boldsymbol{e}$ of weight t_d into two parts, i.e. $\boldsymbol{e} = [\boldsymbol{e}_1 \boldsymbol{e}_2]$ with $\boldsymbol{e}_1$ of length k and $\boldsymbol{e}_2$ of length $n-k$. Now assume that no error happens in the systematic part of the codeword, i.e. $\boldsymbol{e}_1 = \boldsymbol{0}$. The calculation of the syndrome reads*

$$\boldsymbol{s} = \boldsymbol{e} \odot \boldsymbol{H}^{\mathsf{T}} = \boldsymbol{e}_2 \odot \boldsymbol{I}_{n-k} = \boldsymbol{e}_2$$

Thus, $\mathrm{wt}(\boldsymbol{s}) = t_d$ results. Now focus on the case that at least one error happens in the systematic part of the codeword, i.e. $\boldsymbol{e}_1 \neq \boldsymbol{0}$. With this, we calculate the syndrome by

$$\boldsymbol{s} = \boldsymbol{e} \odot \boldsymbol{H}^{\mathsf{T}} = \boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}} \oplus \boldsymbol{e}_2.$$

From encoding $\boldsymbol{e}_1$, we obtain

$$\boldsymbol{e}_1 \odot \boldsymbol{G} = \boldsymbol{e}_1 \odot [\boldsymbol{I}_k \mid \boldsymbol{P}^{\mathsf{T}}] = [\boldsymbol{e}_1 \mid \boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}}].$$

As $\mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{G}) = d \geq 2t_d + 1$ and $\mathrm{wt}(\boldsymbol{e}_1) \leq t_d$, it is known that $\mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}}) \geq t_d + 1$ and with this $\mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}}) \geq \mathrm{wt}(\boldsymbol{e}_2)$ for any $\boldsymbol{e}_2$. Due to the binary addition and as $\mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}}) \geq \mathrm{wt}(\boldsymbol{e}_2)$, we state

$$\mathrm{wt}(\boldsymbol{s}) \geq \mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}}) - \mathrm{wt}(\boldsymbol{e}_2). \tag{6.28}$$

Rewriting the consideration on the weight of a codeword given above, one obtains

$$\mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{G}) = \mathrm{wt}(\boldsymbol{e}_1) + \mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}}) \geq 2t_d + 1. \tag{6.29}$$

Using Equation (6.28) and inserting the lower bound on $\mathrm{wt}(\boldsymbol{e}_1 \odot \boldsymbol{P}^{\mathsf{T}})$ from Equation (6.29), we obtain

$$\begin{aligned} \mathrm{wt}(\boldsymbol{s}) &\geq (2t_d + 1 - \mathrm{wt}(\boldsymbol{e}_1)) - \mathrm{wt}(\boldsymbol{e}_2) \\ &\geq 2t_d + 1 - \mathrm{wt}(\boldsymbol{e}) \\ &\geq t_d + 1. \end{aligned}$$

■

Having these prerequisites at hand, the automorphism decoder is easy to implement. The decoder checks the weight of the syndrome and applies permutations from the automorphism to the received vector $\boldsymbol{y}$ until either $\mathrm{wt}(\boldsymbol{s}) \leq t_d$ is detected or all permutations in $\mathrm{Aut}(\mathcal{C})$ were applied to $\boldsymbol{y}$ without decoding success. In the first case, $\boldsymbol{s} = \boldsymbol{e}_2$ obviously holds in the permuted word. This allows to subtract the error vector from the parity part of the permuted received word and apply the inverse permutation to obtain $\hat{\boldsymbol{x}}$ and subsequently, $\hat{\boldsymbol{u}}$.

In order to keep the complexity of a decoder low, one is interested in finding a small set of permutations which can move a high number of error patterns to the desired positions. To identify sets of this type, we define the function $\mathrm{PD}(\mathrm{Aut}(\mathcal{C}))$ of a code.

Definition 6.3.5 *([Key]) If $\mathcal{C}$ is a t_d-error correcting code with an information bit index set $\mathcal{P}_{\mathrm{syst}}$ and a parity-check index set $\mathcal{P}_{\mathrm{par}}$, then the function $\mathrm{PD}(\mathrm{Aut}(\mathcal{C}))$ selects permutations in the automorphism groups of $\mathcal{C}$ in such a way that the permutations in $\mathrm{PD}(\mathrm{Aut}(\mathcal{C}))$ move every t_d-set of coordinate positions by at least one member of $\mathrm{PD}(\mathrm{Aut}(\mathcal{C}))$ into the parity positions $\mathcal{P}_{\mathrm{par}}$. We refer to the resulting subsets of the elements in the automorphism group as permutation decoding (PD) sets. For $s \leq t_d$, the set $\mathrm{PD}_s(\mathrm{Aut}(\mathcal{C}))$ is a set of permutations of $\mathcal{C}$ such that every b-set of coordinate positions, for all $b \leq s$, is moved by at least one member of this set into $\mathcal{P}_{\mathrm{par}}$. We refer to such a set as an s-PD set.*

We transfer the idea of the PD sets to the stopping set case. There, we are interested in subsets of the automorphism group of the code that have small cardinality and move erasures in such a way that they do not correspond to stopping sets of the current matrix. In order to describe these sets properly, we introduce the idea of automorphism decoding.

6.3.2 Automorphism Decoding

We introduce a permutation decoding approach for reconstructing data transmitted over the BEC. To distinguish this approach from the scheme introduced above, the denomination "automorphism decoding" is used. *For this type of decoding we focus on cyclic codes.*

An automorphism decoder works on a parity-check matrix of standard size $n - k \times n$. When the decoder encounters a stopping set, it makes use of the fact that the stopping set occurs due to the received erasure pattern and the currently used parity-check matrix. With this, the decoder has two options to resume the decoding process. One possibility is to choose another parity-check matrix for the current code which does not contain stopping sets corresponding to the currently observed erasure pattern. A second option is to transform the currently received word using a permutation of the automorphism group of the code.

The first option is motivated by the fact that choosing a different parity-check matrix allows the decoder to resume the decoding process if the erasure pattern does not correspond to a stopping set of this matrix. For the latter option it is easy to see that the result of the permutation is again a valid received word. However, and this is the basic motivation of this algorithm, the

new erasure pattern does not necessarily correspond to a stopping set of the currently used parity-check matrix, as will be shown in Example 6.3.2. Once this is accomplished, the decoder may resume decoding and resolve further erased positions. Notice that the decoder can apply these permutation operations multiple times within the decoding process. Before forwarding the decoding results to the information sink, all permutations need to be undone using the corresponding inverse permutations. The inverse permutations exist for sure as the permutations are chosen from the automorphism group, cf. Appendix B.

Example 6.3.2 *Let us demonstrate the basic principle of the approach with the* $[7, 4, 3]$ *Hamming code*[5] *where three erasures, denoted by ?, occurred during transmission. The rows of the parity-check matrix used in the decoding process,* $\boldsymbol{H}$*, and the received vector* $\boldsymbol{y}$ *are given as follows.*

$$\begin{array}{c|ccccccc} \boldsymbol{y} & 1 & 1 & ? & ? & ? & 0 & 0 \\ \hline \boldsymbol{H}(1,:) & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ \boldsymbol{H}(2,:) & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ \boldsymbol{H}(3,:) & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

None of the parity-check equations of $\boldsymbol{H}$ *can resolve any of the erasures in the currently observed erasure pattern. Using the knowledge that the* $[7, 4, 3]$ *Hamming code is a cyclic code, one applies a cyclic shift to the received word* $\boldsymbol{y}$*, what results in another codeword of the code where* 3 *positions have been erased. Unfortunately this leads to a similar situation. However, after performing a second cyclic shift, denoted by* $\boldsymbol{y}^{(i \to i+2)}$*, the values of the erased symbols can be reconstructed.*

$$\begin{array}{c|ccccccc} \boldsymbol{y}^{(i \to i+2)} & 0 & 0 & 1 & 1 & ? & ? & ? \\ \hline \boldsymbol{H}(1,:) & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ \boldsymbol{H}(2,:) & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ \boldsymbol{H}(3,:) & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Equation $\boldsymbol{H}(1,:)$ *can be used to determine that the value of the first erasure must read* 0*. Alike, the values for the second and the third value result to* 0 *and* 1*, what is deducted from* $\boldsymbol{H}(2,:)$ *and* $\boldsymbol{H}(3,:)$*, respectively. The decoder obtains the result* $\hat{\boldsymbol{x}}^{(i \to i+2)} = 0\,0\,1\,1\,0\,1\,0$*. After undoing all permutation operations, one obtains* $\hat{\boldsymbol{x}} = 1\,1\,0\,1\,0\,0\,0$*.*

From Example 6.3.2 it is straightforward to see that permuting the received word corresponds to decoding on a matrix to which the inverse permutation has been applied.

Example 6.3.3 *(Example 6.3.2 continued) The inverse permutation of a cyclic shift to the right side is a cyclic shift to the left side. The decoding problem can also be solved by the following constellation.*

[5] The code used here is equivalent, but not equal to the one in Example 6.1.1.

$$\begin{array}{c|c} \boldsymbol{y} & \begin{array}{ccccccc} 1 & 1 & ? & ? & ? & 0 & 0 \end{array} \\ \hline \boldsymbol{H}^{(i\rightarrow i-2)}(1,:) & \left[\begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right. \\ \boldsymbol{H}^{(i\rightarrow i-2)}(2,:) & \begin{array}{ccccccc} 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array} \\ \boldsymbol{H}^{(i\rightarrow i-2)}(3,:) & \left.\begin{array}{ccccccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array}$$

This approach uses a transformed parity-check matrix. In this case, no undoing of permutations is needed as the parity-check matrix is not a constituent of the decoder output.

In the following, we concentrate on the solution applying permutations to the codewords. This method can be implemented very efficiently and requires only a low amount of additional memory.

All permutations in the automorphism group of a code can be used for the decoding approach described in the following. We concentrate on the group of cyclic and extended cyclic codes[6], cf. Section 5.1.1, which have a large automorphism group. For the extended cyclic codes, the cyclic property is confined to $n-1$ symbols. W.l.o.g. we fix the position of the the additional encoded symbol to position n. We point out that special code classes like BCH codes of length $n = 2^b - 1$, $b \in \mathbb{N}$ [Hoc59, BRC60], contain the whole *affine group* [MS77, Ch. 8] on the Galois field $\mathbb{F}_{2^a}$ in their automorphism group. The cyclic Golay codes contain the *Mathieu group* [MS77, Ch. 20] in their automorphism group. However, we restrict ourselves on subsets of the automorphism group which are based on the cyclic structure. Simulation results show that already with this partial information, the automorphism decoding approach obtains significant performance improvements and approaches the performance of an ML decoder surprisingly closely. Before discussing these permutations, we introduce a variable h, which depends on n and denotes the cardinality of the cyclotomic coset [MS77] of the n-th roots of unity that contains one. We also make use of the operator $\mathrm{mod}^*(\cdot)$, which identifies a non-standard modulo-function where multiples of the modulus result in the modulus instead of zero. This function is defined as $i \bmod^* n = i \bmod n + n\delta[(n-i) \bmod n]$, $i \in \mathbb{N}$, $n \in \mathbb{N}$. The function $\delta[j]$ is the discrete unit impulse [GRS01] and reads

$$\delta[j] = \begin{cases} 1 & \text{if} \quad j = 0 \\ 0 & \text{if} \quad j \in \mathbb{Z}\backslash\{0\} \end{cases}.$$

The considered permutations are divided into two classes.

P_1: The class of cyclic permutations $\alpha^0, \alpha^1, \ldots, \alpha^{n-1}$ to the right, where

$$\begin{aligned} \alpha : i &\rightarrow (i+1) \bmod^* n,\ i = 1, \ldots, n, \\ \alpha(\boldsymbol{x}) &= [x_n, x_1, x_2, \ldots, x_{n-1}]. \end{aligned}$$

[6]Note that the approach works for the larger class of codes with a parity-check matrix of cyclic form. However, as all examples investigated in this chapter are (extended)-cyclic codes, we focus on this subset.

P_2: The class of permutations $\beta^0, \beta^1, \ldots, \beta^{h-1}$, where

$$\begin{aligned} \beta : i &\rightarrow (2 \cdot (i-1)) + 1 \bmod^* n,\ i = 1, \ldots, n, \\ \beta(\boldsymbol{x}) &= [x_1\, x_{(n+3)/2}\, x_2\, \ldots x_{(n+1)/2}]. \end{aligned} \tag{6.30}$$

We follow the notation for permutations which was introduced in Definition 5.1.10. Above, all subscripts are taken modulo $n+1$. When using extended cyclic codes, these permutations are only applied to the cyclic part of the code.

Theorem 6.3.6 *A binary linear cyclic $[n, k, d]$ code $\mathcal{C}$ has odd code length $n = 2b - 1$, $b \in \mathbb{N}$ and its automorphism group contains both P_1 and P_2.*

Proof 6.3.7 *Cyclic codes are constructed in a Galois Field $\mathbb{F}_q$ such that n divides $2^{\mathrm{ld}(q)}-1$ [MS77, Ch. 7]. Thus, n is odd, $n = 2b - 1$, $b \in \mathbb{N}$. The definition of a cyclic code, cf. Definition 5.1.4, ensures that permutation P_1 is in the automorphism group of the code. Let us write the set of codewords of $\mathcal{C}$ as a polynomial in ξ. This results to*

$$x(\xi) = \bigoplus_{j=0}^{n-1} x_{j+1}\xi^j.$$

Applying the permutation $\beta : \ i \rightarrow 2i$ to the vector $\boldsymbol{x} = [x_1, \ldots, x_n]$, corresponding to the polynomial $x(\xi)$, one obtains a resulting vector which corresponds to $x(\xi^2)$. Let us now show that $x(\xi^2) = (x(\xi))^2$ holds when operating in a Galois field of characteristic two.

$$\left(\bigoplus_{j=0}^{n-1} x_{j+1}\xi^j\right)^2 = \bigoplus_{j=0}^{n-1} x_{j+1}^2\xi^{2j} + \underbrace{\bigoplus_{j=0}^{n-1}\bigoplus_{j'=0, j'\neq j}^{n-1} x_{j+1}x_{j'+1}\xi^{j+j'}}_{\bigoplus_{j=0}^{n-1} 2x_{j+1}x_{j'+1}\xi^{j+j'},\ \forall j' \neq j} = \bigoplus_{j=0}^{n-1} x_{j+1}\xi^{2j}.$$

In words, all mixed terms vanish, what is due to the fact that we operate on a Galois field of characteristic two. Thus, we can rewrite $x(\xi^2)$ as a linear combination of cyclic shifts of $x(\xi)$,

$$x(\xi^2) = x(\xi)\left(\bigoplus_{j=0}^{n-1} x_{i+1}\xi^j\right). \tag{6.31}$$

With this, $x(\xi^2)$ is a codeword of the current code. ∎

Figure 6.3 illustrates the permutation operations used in the described decoding architectures. For the illustration we use the more general concept of extended cyclic codes. However, the permutations only apply to the cyclic part of the code, i.e. to positions 1 to $n-1$. Observe in

Figure 6.3, that the check nodes and edges in the graph of an (extended) cyclic code are fixed, while the variable nodes are permuted according to the permutations P_1 and P_2.

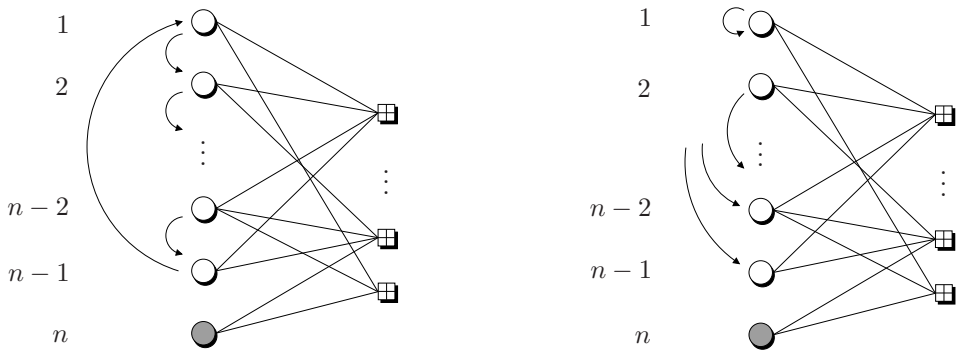


Figure 6.3: Permutation from the class P_1 (left-hand side figure) and P_2 (right-hand side figure) for extended cyclic codes. The position of the overall parity bit is marked by grey color.

The stopping automorphism decoder (SAD) uses the permutations of the automorphism group to decode cyclic codes although stopping sets occur. As a first option, we resume decoding in case of a stopping set by applying permutations of P_1 and denote this approach by SAD-1. A second approach is denoted by SAD-2. Here, we additionally use permutations of P_2 if all permutations of type P_1 were tried without success. The decoder stops once it has reached the maximum number of iterations, found a valid codeword, or tested all possible permutations on a single stopping set. A description in pseudocode of SAD-2 is given in Algorithm 6.1. Once the decoder fails to decode at least one erasure and runs out of valid permutations, it terminates the iterative decoding process and has to guess all remaining erasures. Algorithm 6.2 shows the post-processing step which guesses all unrecovered symbols independently of each other.

Smart Guessing for Lowering the Frame Error Rate

If at least one erasure remains unresolved in the decoding process, the decoder has to guess its value. As stated in Algorithm 6.2, these guesses are conducted independently of each other. A typical consequence are frames which do not represent a codeword, what is a conceptual disadvantage and prohibits the decoder from reaching the performance of an ML decoder in terms of the FER. The latter decoder could consider a list of possible codewords fitting to the already known symbol positions. However, a BP decoder does not have this information readily available.

In order to solve this problem, we introduce an approach termed smart guessing (SG) [Mil06]. If the maximum number of iterations is not reached, the decoder is allowed to guess *a single* erased position and try to resume the decoding process. This position can be chosen randomly, however a lower number of tries is required on the average if positions with a high column degree in the currently used parity-check matrix are preferred as candidates. This small addendum allows for a significantly improved FER. The results presented in Section 6.4 show that the BER-performance

Algorithm 6.1 SAD-2 decoding.

Input: $\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, h, n
Output: $\hat{\boldsymbol{x}}$, success_flag
Function call: [$\hat{\boldsymbol{x}}$, success_flag]=sad_2_decode($\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, h, n)

```
permutations_tested_P1 = 0 /* All permutations from P_1 tested? */
permutations_tested_P2 = 0 /* All permutations from P_2 tested? */
while I < I_max and permutations_tested_P2 < h and success_flag == 0 do
   /* Perform BP iteration */
   [x̂, success_flag]=bp_decode(y, H, 1)
   /* Check if the iteration could resolve any erasures */
   if x̂ == y then
      /* Apply permutation to current input vector */
      if permutations_tested_P1 < n then
         y = permutation_next_P1(y)
         permutations_tested_P1 = permutations_tested_P1 + 1
      else if permutations_tested_P2 < h then
         y = permutation_next_P2(y)
         permutations_tested_P2 = permutations_tested_P2 + 1
      end if
   else
      y = x̂ /* Progress in Decoding, prepare for next iteration */
      I = I + 1
      /* Reset function permutation_next_P1(·) and permutation_next_P2(·) */
      permutation_next_P1().reset()
      permutation_next_P2().reset()
      permutations_tested_P1 = 0 /* Reset counter after iteration */
      permutations_tested_P2 = 0 /* Reset counter after iteration */
   end if
end while
```

Algorithm 6.2 Post processing for SAD-2 decoding. Erased positions are denoted by '?'.

Input: $\hat{\boldsymbol{x}}$
Output: $\hat{\boldsymbol{x}}$
Function call: $\hat{\boldsymbol{x}}$=post_proc_guessing($\hat{\boldsymbol{x}}$)

```
/* Guess non-recovered positions */
i_? = find(x̂ ==?)
x(i_?) = rand({0, 1})
```

is improved as well, but not in such a significant manner. This was expected, as the smart guessing method was tailored to overcome the drawback of guessing the unrecovered positions independently, which eminently exists for the FER performance. In order to implement smart guessing, one needs to replace Algorithm 6.2 by Algorithm 6.3. In order to keep the presentation simple, the positions at which guessing is performed are chosen randomly in Algorithm 6.3.

Algorithm 6.3 Smart-guessing post-processing for SAD-2. Erased positions are denoted by '?'.
Input: $I_{\max}$, $\hat{\boldsymbol{x}}$, $\boldsymbol{H}$, h, n
Output: $\hat{\boldsymbol{x}}$
Function call: $\hat{\boldsymbol{x}}$=post_proc_smart_guessing($\hat{\boldsymbol{x}}$, $\boldsymbol{H}$, $I_{\max}$, h, n)
1: **while** $I < I_{\max}$ **and** success_flag == 0 **do**
2: $\boldsymbol{i}_? = \text{find}(\hat{\boldsymbol{x}} == ?)$
3: $i_?^* = \text{rand}(\{i_?(1), \ldots, i_?(\text{end})\})$
4: $\hat{\boldsymbol{x}}(i_?^*) = \text{rand}(\{0, 1\})$
5: $\boldsymbol{y} = \hat{\boldsymbol{x}}$
6: $[\hat{\boldsymbol{x}}, \text{success\_flag}]$=sad_2_decode($\boldsymbol{y}$, $\boldsymbol{H}$, 1, h, n)
7: **end while**

The Automorphism Redundancy

Similar to the permutation decoding sets, cf. Definition 6.3.5, we are interested in small sets containing permutations which allow to remove a high number of stopping sets. We refer to these sets as stopping automorphism group decoding sets, to be defined shortly. We compare the sizes of these sets to the sizes of permutation decoding sets and relate them to the stopping redundancy hierarchy of a code.

Definition 6.3.8 *Let $\boldsymbol{H}$ be a parity-check matrix of an error-correcting code $\mathcal{C}$ with minimum distance d and $n-k$ rows. A set $\text{SAD}(\text{Aut}(\mathcal{C}), \boldsymbol{H})$ of $\mathcal{C}$ for a given $\boldsymbol{H}$ is the smallest subset of permutations in $\text{Aut}(\mathcal{C})$ such that every b-set of coordinate positions, $1 \leq b \leq d-1$, is moved by at least one member of $\text{SAD}(\text{Aut}(\mathcal{C}), \boldsymbol{H})$ into a set of positions that do not correspond to a stopping set of $\boldsymbol{H}$. We refer to this set of automorphisms as a stopping automorphism group decoding (SAD) set. To relate this entity to the stopping redundancy hierarchy, let $s \leq d-1$. A set $\text{SAD}_s(\text{Aut}(\mathcal{C}), \boldsymbol{H})$-set is the smallest set of automorphisms in the automorphism group of $\mathcal{C}$ such that every b-set of coordinate positions, for all $b \leq s$, is moved by at least one member of $\text{SAD}_s(\text{Aut}(\mathcal{C}), \boldsymbol{H})$ into positions that do not correspond to a stopping set in $\boldsymbol{H}$. We refer to $\text{SAD}_s(\text{Aut}(\mathcal{C}), \boldsymbol{H})$ as an s-SAD set. Without loss of generality, we assume that a set $\text{SAD}_s(\text{Aut}(\mathcal{C}), \boldsymbol{H})$ contains the identity permutation. For a given code $\mathcal{C}$, we also define*

$$\begin{aligned} S_s^\star(\mathcal{C}) &= \min_{\boldsymbol{H}(\mathcal{C})} |\text{SAD}_s(\text{Aut}(\mathcal{C}), \boldsymbol{H})|, \\ S^\star(\mathcal{C}) &= \min_{\boldsymbol{H}(\mathcal{C})} |\text{SAD}(\text{Aut}(\mathcal{C}), \boldsymbol{H})|, \end{aligned}$$

and refer to $S_s^\star(\mathcal{C})$ and $S^\star(\mathcal{C})$ as to the s-automorphism redundancy *and* automorphism redundancy *of $\mathcal{C}$, respectively.*

Notice that an s-SAD does not necessarily have to exist for a given choice of s. This is due to the fact that the automorphism group is limited and a permutation moving all selections of s variables into positions not corresponding to stopping sets may not exist for the matrix. Whenever an s-PD set and an s-SAD set exist, there exists a relation concerning their sizes.

Moreover, a relation to one corresponding element of the stopping redundancy hierarchy exists. Both relations are discussed in the following Lemma 6.3.9 and Lemma 6.3.11.

Lemma 6.3.9 *The statement*

$$S_s^\star(\mathcal{C}) \leq |\mathrm{PD}_s(\mathrm{Aut}(\mathcal{C}))|$$

holds for all $1 \leq s \leq d-1$.

This result states that the task of moving s erasures to positions not corresponding to stopping sets is more easy than moving s errors to parity positions.

Proof 6.3.10 *Given is a binary parity-check matrix and binary generator matrix of systematic form, i.e.* $\boldsymbol{H} = [\boldsymbol{A} \mid \boldsymbol{I}_{n-k}]$ *and* $\boldsymbol{G} = [\boldsymbol{I}_k \mid \boldsymbol{A}^\mathsf{T}]$. *It is easy to see that* $\mathcal{P}_{\mathrm{par}} = \{k+1, \ldots, n\}$ *and with this choice, no stopping sets of size* $\sigma > 1$ *exist if the restriction of* $\boldsymbol{H}$ *to* $\mathcal{P}_{\mathrm{par}}$ *is considered. The singleton bound asserts for any linear code* $d-1 \leq n-k$ *and thus guarantees that all stopping sets of size up to* $d-1$ *can be solved by moving them to* $\mathcal{P}_{\mathrm{par}}$. *However, depending on the structure of the matrix, there exist more sets of positions that do not correspond to stopping sets. The additional flexibility can only decrease the number of required permutations.* ■

The s-automorphism redundancy can be used to find an upper bound on the $s+1$-th element of the stopping redundancy hierarchy.

Lemma 6.3.11 *Let* $\mathcal{C}$ *be an* $[n, k, d]$ *code and assume that an s-SAD set exists. Then*

$$\rho_{s+1}(\mathcal{C}) \leq (n-k) \cdot S_s^\star(\mathcal{C})$$

holds for $1 \leq s \leq d-1$.

Proof 6.3.12 *The proof of Lemma 6.3.11 follows from the fact that each permutation applied to the current decoding matrix adds at most (n-k) "virtual" redundant checks. The qualifier 'at most' is due to the possibility of adding existing checks again.* ■

Remark 6.3.13 *The result of Lemma 6.3.11 can be understood in the following way. Whenever a permutation to the codeword is applied, the decoding capability is increased. This is as a virtual increase of the number of parity checks. Note in this context that the original equations are not removed but "suspended" for a given number of iterations. This is due to the fact that whenever an erasure is recovered, the set of possible permutations is reinitialized and the original parity-check information can be restored by permutation operations.*

There exists one class of codes for which the existence of SAD sets of certain sizes is given for sure. This is the class of codes with *transitive* automorphism groups, described below.

Definition 6.3.14 *A group Ξ of permutations of the symbols $\{1, 2, \ldots, n\}$ is transitive if for any two symbols i, j there exists a permutation $\zeta \in \Xi$ such that $\zeta : i \to j$. A group is said to be ℓ-fold transitive if for any two collections of ℓ distinct numbers $i_1, \ldots, i_\ell \in \{1, \ldots, n\}$ and $j_1, \ldots, j_\ell \in \{1, \ldots, n\}$, there exists a $\zeta \in \Xi$ such that $\zeta : i_1 \to j_1, \ldots, \zeta : i_\ell \to j_\ell$.*

Lemma 6.3.15 *Let $\mathcal{C}$ be a code with an s-transitive automorphism group. Then there exist b-SAD sets of $\mathcal{C}$ for all $b \leq s$. It is assumed that $s \leq n - k$.*

Proof 6.3.16 *Let $\boldsymbol{H}$ be of the form $[\boldsymbol{A} \mid \boldsymbol{I}]$. Clearly, the positions of $\boldsymbol{H}$ indexed by $k+1$ through n are free of stopping sets of size $s \leq n - k$. Since the automorphism group of $\mathcal{C}$ is s-transitive, any collection of not more than s coordinates in $\{1, \ldots, n\}$ is moved by some element of $\mathrm{Aut}(\mathcal{C})$ into the positions indexed by $k + 1$ to n. Consequently, the automorphism group itself represents a (possibly non-minimal) s-SAD set.* ∎

6.4 Performance Results

Finding s-SAD sets of codes is a very complicated task. In the following, we focus on deriving bounds on the size of these sets for specific examples of codes. We present results which allow to compare the proposed automorphism decoding approach with standard schemes and known schemes using redundancy. Simulation results are presented for various algebraic codes. Prior to this, we investigate upper bounds on the stopping redundancy hierarchy, derived from several decoding methods for the $[24, 12, 8]$ extended Golay code. Also for this code we count the number of unresolved stopping sets and dead-end sets and relate these results for different matrices and schemes to each other. The number of unresolved stopping sets of size σ when using the parity-check matrix $\boldsymbol{H}$ is denoted by $\left|\overline{\Sigma}_\sigma(\boldsymbol{H})\right|$ with $\left|\overline{\Sigma}_\sigma(\boldsymbol{H})\right| = \binom{n}{\sigma} - |\Sigma_\sigma(\boldsymbol{H})|$, $\forall\sigma$. For brevity we also use the term "number of stopping sets". We choose the extended Golay code for a detailed investigation due to its shortness, the rich mathematical structure and the fact that it is often used as an example in literature [SV06, HS07].

6.4.1 Decoding Performance of the $[24, 12, 8]$ Extended Golay Code

In the following, we discuss parity-check matrix representations and (partly general) redundant decoding schemes on the basis of the $[24, 12, 8]$ extended Golay code. As we are interested in BP decoding, what relies on a sparse parity-check matrix, we try to fulfill this requirement as good as possible by using only codewords of minimum weight in the parity-check matrix.

The $[24, 12, 8]$ extended Golay code is self-dual and its weight distribution can easily be found analytically [MS77] or by computer search techniques. The 4096 codewords are distributed according to the weight enumerator

$$\mathrm{A}(x) = x^0 + 759x^8 + 2576x^{12} + 759x^{16} + x^{24}.$$

We use only the 759 codewords of weight 8 to construct parity-check matrices for this code. It turns out that these codewords can be partitioned into 33 COGs. Let us first review existing approaches for decoding the $[24, 12, 8]$ extended Golay code and then compare them with the approach proposed by us.

Wolfmann [Wol83] suggested a set of 14 permutations suitable for permutation decoding (see Section 6.3.1) together with a parity-check matrix, henceforth denoted by $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{W}}$,

$$\boldsymbol{H}_{\text{Golay},[24,12,8],\text{W}} = \begin{bmatrix} I_3 & A & A^2 & A^4 \\ A & I_3 & A^4 & A^2 \\ A^2 & A^4 & I_3 & A \\ A^4 & A^2 & A & I_3 \end{bmatrix}, \text{where } A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

For the corresponding permutations, to be introduced shortly, no easy mapping functions exist. Hence, we denote the automorphisms in standard cycle form [MS77]. This notation is explained in the following remark.

Remark 6.4.1 *The standard cycle form shows the partitioning of the permutation into cycles, each of them marked by enclosing round brackets. In order to perform the permutation, one cyclic shift from each of the cycles needs to be conducted.*

Example 6.4.1 *Let a permutation be given in standard cycle form by*

$$(1, 2, 3)(4)(6)(5, 7).$$

The application of this permutation permutes the positions such that the variable on position 1 *is shifted to position* 2 *while the variable on position* 2 *is shifted to position* 3*. The variable on position* 3 *is shifted to position* 1*. The same is done for the indices in the second pair of brackets, i.e. the variable on position* 4 *stays put. Together with the two remaining cycles, we subsume*

$$[x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7] \rightarrow [x_3\, x_1\, x_2\, x_4\, x_7\, x_6\, x_5].$$

The permutations suggested by Wolfmann [Wol83] have the form $\alpha_{\text{W},1}^i \times \alpha_{\text{W},2}^j$, $i = 0, 1$, $j = 0, 1, \ldots, 5, 6$, where

$\alpha_{\text{W},1}$:

$$(1, 13)(2, 14)(3, 15)(4, 16)\ldots(11, 23)(12, 24),$$

$\alpha_{\text{W},2}$:

$$(1)(2)(3)(4, 7, 16, 10, 22, 19, 13)(5, 8, 17, 11, 23, 20, 14)(6, 9, 18, 12, 24, 21, 15).$$

We use the matrix $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{W}}$ along with the permutations for automorphism decoding. For convenient notation, we define a stopping automorphism decoder which uses the permutations of

type $\alpha_{\mathrm{W},1}$ and $\alpha_{\mathrm{W},2}$ by SAD-W. This decoder mimics the operations performed by SAD-2, but uses $\alpha_{\mathrm{W},1}$ and $\alpha_{\mathrm{W},2}$. Wolfmann has used these permutation to show that $|\mathrm{PD}_s(\mathrm{Aut}(\mathcal{C}))| = 14$ [Wol83] for the $[24, 12, 8]$ extended Golay code. We investigate this set of permutations with respect to its capabilities in SAD decoding.

Following Lemma 6.3.9 and Lemma 6.3.11 we state

$$\rho_i(\mathcal{C}) \leq 12 \times 14 = 168, i \leq 8.$$

This bound is very loose compared to the constructive upper bound given by the matrix shown in Equation (6.32). This matrix, referred to as $\boldsymbol{H}_{\mathrm{Golay},[24,12,8],1}$, obtains $\rho_6(\mathcal{C}) = 21$ and was found by computer search. Also, upper bounds on $\rho_8(\mathrm{Golay}, [24, 12, 8])$ with significantly less than 168 rows exist, as will be shown shortly.

$$\boldsymbol{H}_{\mathrm{Golay},[24,12,8],1} = \left[\begin{array}{cccccccccccccccccccccccc}
1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&1&0&0&0&0&0\\
0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&1&0&0&0\\
0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&1&0&0\\
0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&1&0\\
1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0\\
1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&0\\
0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0\\
0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0\\
0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0\\
1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&1&0\\
1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0&0\\
0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&1&0\\
1&0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&0\\
0&1&0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0&0\\
0&0&1&0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&1&0\\
1&0&0&1&0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&1&0\\
1&1&0&0&1&0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0&0\\
0&1&1&0&0&1&0&1&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0&0\\
1&0&1&0&1&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right]. \tag{6.32}$$

Schwartz and Vardy [SV06] as well as Han and Siegel [HS07] proposed matrices for the $[24, 12, 8]$ extended Golay code which mark upper bounds on $\rho_8(\mathrm{Golay}, [24, 12, 8])$. These matrices were found by computer search and both have 34 rows, thus $\rho_8(\mathrm{Golay}, [24, 12, 8]) \leq 34$. We denote these matrices by $\boldsymbol{H}_{\mathrm{Golay},[24,12,8],\mathrm{SV}}$ and $\boldsymbol{H}_{\mathrm{Golay},[24,12,8],\mathrm{HS}}$, respectively. For decoding with the SAD algorithm, we exclusively use the matrix $\boldsymbol{H}_{\mathrm{Golay},[24,12,8],\mathrm{SAD}}$, cf. Equation (6.33). None of its rows can be transformed into each other by a permutation of type P_1. A computer search algorithm was used to keep the number of rows that can be transformed into each other by means of P_2 low.

$$\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SAD}} = \begin{bmatrix}
1&1&1&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1\\
1&1&0&0&0&0&1&0&0&0&0&1&0&0&1&1&1&0&0&0&0&0&0&1\\
1&1&0&1&0&0&1&0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1\\
1&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&\\
1&1&0&0&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1\\
1&1&0&1&0&0&0&1&0&1&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
0&1&1&0&0&0&1&0&0&0&0&1&0&1&0&0&0&1&0&0&1&0&0&1\\
1&1&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1\\
1&1&1&1&0&1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1\\
1&1&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&1&0&1\\
0&0&1&1&0&0&1&0&1&0&0&1&0&0&1&0&0&0&0&1&0&0&0&1\\
0&0&1&1&0&1&0&1&0&0&1&1&0&0&1&0&0&0&0&0&0&0&1&0
\end{bmatrix}. \tag{6.33}$$

This matrix was found by computer search and is very amenable for the automorphism decoding approach.

Let us now compare the number of stopping sets when using a variety of matrices and decoders. First, we discuss the matrix suggested by Wolfmann $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{W}}$ and decode it by both standard BP and the SAD-W decoder. Second, the redundant matrices $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SV}}$ and $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{HS}}$ known from literature are considered. We decode these matrices with standard BP decoding. These results are related to the proposed matrix $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SAD}}$. This matrix is decoded on a standard BP decoder and the SAD-1 decoder, respectively. For reference, we present the results of an ML decoder performing full search. The resulting numbers of stopping sets are given in Table 6.1. Note that for an ML decoder stopping sets are not defined, as no parity-check matrix is specified. The ML decoder is only influenced by the presence of ML-dead-end sets, i.e. codewords with a completely erased support, and hence this decoder is only listed in Table 6.2. Also in this table, we show the number of dead-end sets of decoding constellations which compare favorably in Table 6.1.

Table 6.1 shows that the number of stopping sets when decoding with $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SAD}}$ and the SAD-1 decoder is significantly lower than for standard BP with $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SAD}}$. The latter approach also outperforms the redundant matrices $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SV}}$ and $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{HS}}$ decoded with standard BP. The number of stopping sets when using the SAD-W decoder running on $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{W}}$ and the SAD-1 decoder running on $\boldsymbol{H}_{\text{Golay},[24,12,8],\text{SAD}}$ are comparable. For this reason, let us also compare the number of dead-end sets for both types and relate the result to the optimal ML decoder, for which we count ML-dead-end sets.

Table 6.2 shows that the number of dead-end set for the SAD approach and the number of ML-dead-end sets (ML decoder) equal up to a pattern size of 11, whereas the approach from [Wol83] shows a significantly higher number of dead-end sets. Simulation results (cf. Section 6.4.2) show that this corresponds to a measurable advantage in terms of the BER and FER performance. It is worth pointing out that the number of stopping sets of size $\sigma = 9$ and the number of dead-end sets of size 9 for the approach in [Wol83] is not contradicting. The fact that the number of

σ	Number of unresolved stopping sets, $\lvert\overline{\Sigma}_\sigma(\boldsymbol{H})\rvert$		
	$\boldsymbol{H}_{\text{Golay,[24,12,8],SAD}}$, BP	$\boldsymbol{H}_{\text{Golay,[24,12,8],SAD}}$, SAD-1	$\boldsymbol{H}_{\text{Golay,[24,12,8],SV}}$, BP
3	7	0	0
4	122	0	0
5	1351	0	0
6	10677	0	0
7	58206	0	0
8	220828	759	3598
9	600590	1932	57956
10	1215705	14904	340961
11	1895016	79028	985791
12	2338064	336858	1712100
	$\boldsymbol{H}_{\text{Golay,[24,12,8],HS}}$, BP	$\boldsymbol{H}_{\text{Golay,[24,12,8],W}}$, BP	$\boldsymbol{H}_{\text{Golay,[24,12,8],W}}$, SAD-W
3	0	0	0
4	0	75	0
5	0	1332	0
6	0	11622	0
7	0	62736	0
8	3284	230760	759
9	53706	613112	569
10	329171	1225800	11073
11	974348	1900848	92682
12	1706657	2340709	501004

Table 6.1: Stopping sets for the $[24, 12, 8]$ extended Golay code.

σ	Number of dead-end sets / ML-dead-end sets		
	$\boldsymbol{H}_{\text{Golay,[24,12,8],SAD}}$, SAD-1	$\boldsymbol{H}_{\text{Golay,[24,12,8],W}}$, SAD-W	ML decoding
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	759	759	759
9	12144	12158	12144
10	91080	93477	91080
11	425040	481764	425040
12	1322178	1547590	1313116

Table 6.2: Dead-end sets and ML-dead-end sets for the $[24, 12, 8]$ Golay code.

stopping sets of size $\sigma = 9$ is lower than the number of size $\sigma = 8$ and the converse behavior is observed for the number of dead-end sets shows that the stopping sets of size 9 do not tend to include stopping sets of size 8.

6.4.2 Simulation Results for the $[24, 12, 8]$ Extended Golay Code

We present simulation results for $[24, 12, 8]$ extended Golay code. Figure 6.4 shows the resulting BER and FER, as well as the average number of iterations performed by different decoding approaches. Simulations have shown that 50 iterations are sufficient for decoding one codeword, and hence this number was used to generate the simulation results. In fact, a significantly lower number of iterations is required on the average[7]. It is to observe that both SAD-1 and SAD-2 achieve excellent performance results. Decoding with SAD-2 is slightly better and both outperform the approach using the automorphisms from [Wol83]. The vertical bar in the performance plots marks the erasure probability for which the decoding complexity of all approaches, compared by means of the performed iterations, is comparable. In other words, the performance improvement measured for erasure probabilities smaller than the one marked by this bar comes at very little cost. Another observation from Figure 6.4 is worth pointing out. The performance of the ML decoder is not attained if the FER is the error rate of interest. The reason for the drawback in the FER-performance is the post-processing step which guesses the unresolved erasures. The smart guessing method can be used to close this performance gap, what is shown exemplary for the $[63, 39, 9]$ BCH code.

6.4.3 Simulation Results for the $[63, 39, 9]$ BCH Code

Figure 6.5 presents simulation results for the $[63, 39, 9]$ BCH code. At most 50 iterations were performed to decode one codeword. Due to its dimension and co-dimension we cannot provide performance results obtained by a full search but present a tight upper bound found by means of the union bound, cf. Section 5.1.4. We observe that the performance using the permutations in the automorphism group of the code comes close to the optimal performance. The close-up view in Figure 6.5 demonstrates the benefit how the smart guessing method can be used to improve the FER-performance of a decoder. In this example, we have used the SAD-1 decoder together with smart guessing approach to approach the performance of the ML decoder in terms of the frame error rate.

[7] We remark that BP decoding on the BEC in general requires a significantly lower number of iterations for decoding when comparing to BP decoding on the AWGN channel. Reasons for this include the fact that no received values lead the decoder into the wrong direction and no contradicting information is sent to a variable node.

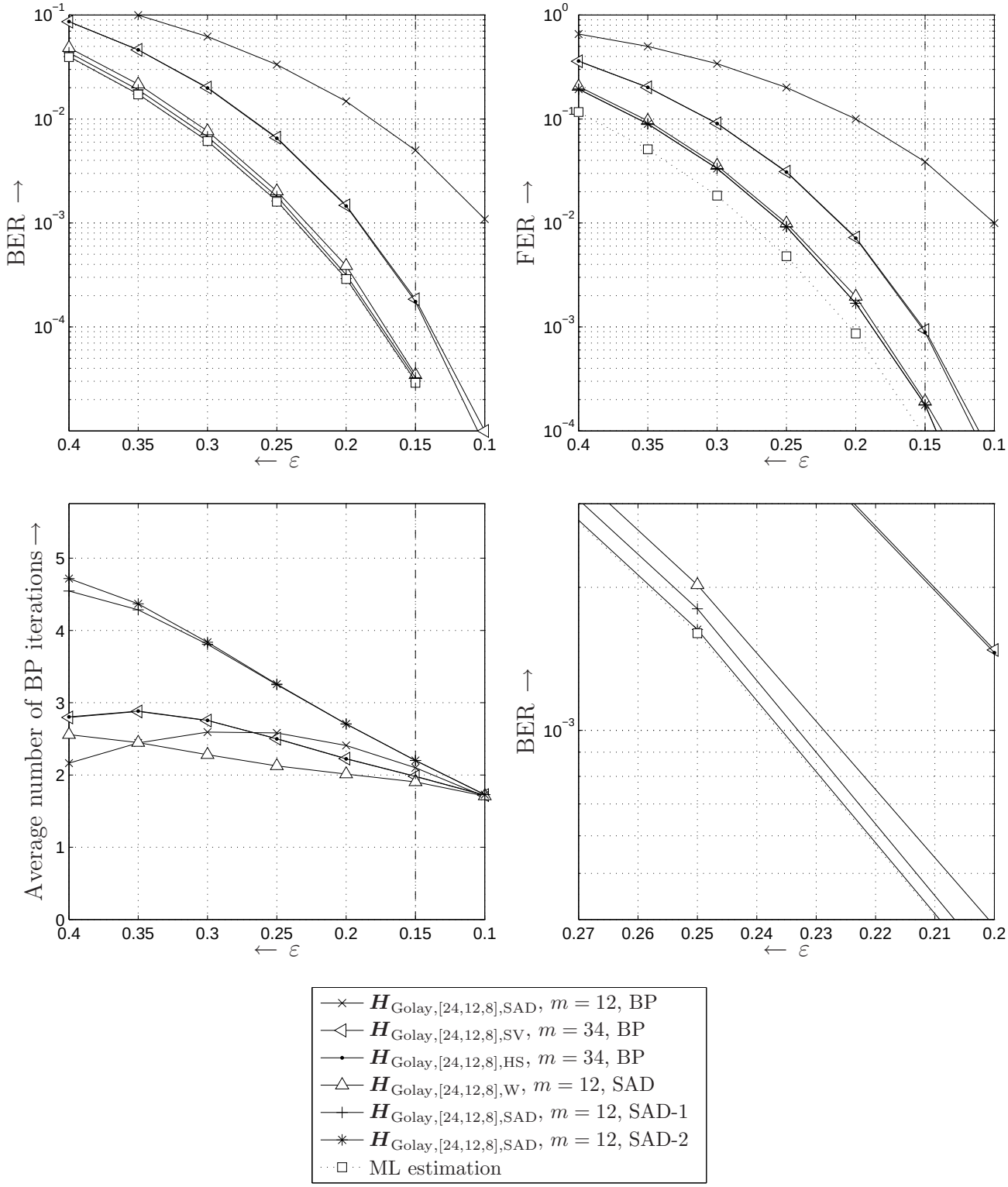


Figure 6.4: Performance of the $[24, 12, 8]$ extended Golay code. Shown are the BER, the FER, the average number of iterations required, and a close up view of the BER.

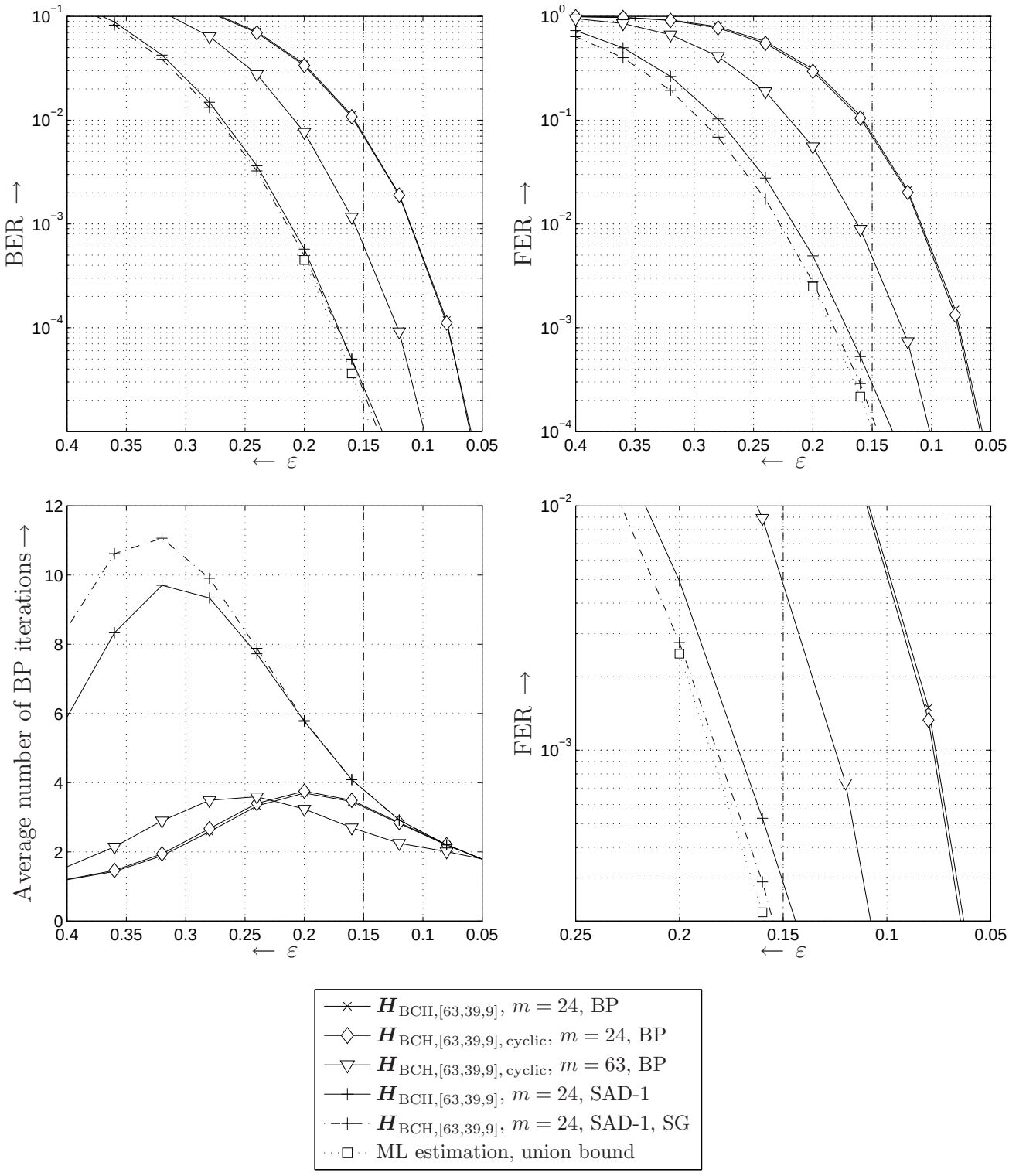


Figure 6.5: Performance of the $[63, 39, 9]$ BCH code. Shown are the BER, the FER, the average number of iterations required, and a close up view of the FER.

Chapter 7

Improved Iterative Decoding on the AWGN Channel: MBBP

It is desirable to transfer the performance improvements obtained by redundant decoding for the BEC to the AWGN channel. We intend to obtain a comparable performance improvement for codes of length $n \leq 1000$. We first discuss the difficulties of redundant BP decoding in connection with a soft-output channel such as the AWGN channel. Subsequently, we introduce methods which mitigate these difficulties and hence allow for performance improvements based on redundant code descriptions.

7.1 Redundant Decoding on the AWGN Channel

Several problems arise when transferring redundant decoding as used for the BEC to the AWGN channel. First, it is not straightforward to design a decoding scheme that allows to generate a redundant row *on request*. Unlike in the BEC case, it is difficult to obtain clear signs indicating that the decoding process has stalled. The theory of trapping sets [LHMH06, Lae09] studies scenarios in which the decoder is unable to proceed. Even if trapping sets occur it cannot be guaranteed that the messages in the decoder are fixed over several iterations. Another reason for decoding failures is given if the decoder moves to an erroneous decision, what is even harder to detect.

In order to provide the decoder with redundant code information, it is still possible to deploy a highly redundant matrix which is used during the whole decoding process. This approach would require a significantly higher computational amount for each iteration but could lead to better performance results. Second, BP decoding on the AWGN channel is sensitive to the local girth distribution, cf. Section 5.1.2. Computer simulations show that appending a high amount of redundant rows to an existing parity-check matrix actually impairs the decoding performance of a BP decoder instead of improving it. We present an approach which mitigates this problem and allows for well-performing redundant decoding of data signaled over the AWGN channel. In

a nutshell, this approach partitions the parity-check information to multiple decoders. Prior to this, we review work already published in this field.

Decoding with multiple decoding attempts can be considered as a simple form of redundancy. Results in this direction were already presented in 1972 by David Chase [Cha72] and later by Fossorier [FL95] as well as Lucas [LBB98]. Fossorier extended the scope of his approach to LDPC codes in [Fos01]. The applicability of this scheme is limited as a costly post-processing step for each received word is required. Further work using redundancy in connection with BP decoding has been done in [ADP02]. There, an approach using several parity-check matrices was briefly discussed and applied to the [24,12,8] extended Golay code. Starting 2005, this topic drew more and more attention. Several authors proposed *adaptive BP algorithms* [KTJF05, JN06], which iteratively optimize the matrix representation during the decoding process. This kind of scheme needs to perform a matrix reduction after each iteration and hence extends the duration of the decoding process significantly.

The random redundant decoding (RRD) [HC06, HC08] algorithm does not require this type of processing and obtains very promising results. This is accomplished by serially altering the applied matrix representation within the decoding process. The main difference between the RRD algorithm and the scheme introduced in this thesis is that RRD operates in a *serial* fashion in terms of periodically permuting the received word, whereas the approach to be presented here works in parallel mode. The possibility to parallelize the decoding process is a very prominent advantage of the latter approach. In contrast to RRD, the completion of the decoding process is not delayed by the use of redundancy. Also, the RRD algorithm uses message scaling between iterations to slow down the convergence of the decoding process. The so called "damping factor" is estimated by simulation and additionally delays the decoding process. Recently published results on this algorithm [HC08] confirm that the decoding algorithm presented in this thesis outperforms the RRD algorithm, cf. Section 7.6.2. Furthermore, the study in [HC06, HC08] is limited to algebraic codes of length up to $n = 63$.

7.2 Multiple-Bases Belief-Propagation Decoding

In this section we describe an approach which uses redundant parity-check equations without impairing the local girth distribution. We introduce several variations of this algorithm, each of them showing advantages in different fields. As a first step, we introduce the most straightforward approach and demonstrate its benefit by a case study. Subsequently, we present the remaining approaches and conduct a performance comparison. The algorithm deploys a set of non-equivalent full-rank parity-check matrices and decodes each received word $\boldsymbol{y}$ with all of these matrices. For each matrix, a separate BP decoder is used. The different matrices allow for decoding of different error patterns, what results in "decoder diversity". As all of these matrices form a *basis* of the dual code, we refer to this approach as multiple-bases belief-propagation (MBBP) [HHLM07, HHML07, HHML09] decoding. Having the decoding outputs of these decoders ready

at hand, it remains to be clarified which candidate is forwarded to the information sink. The most self-evident and best-performing solution for this task is implemented in the primary MBBP approach denoted by *MBBP-NX-S*.

7.2.1 The MBBP-NX-S Algorithm

The first algorithm introduced is the MBBP-NX-S algorithm. The qualifier NX points out that there is no information exchange between the decoders during runtime and the letter S indicates that this variation is the standard approach.

MBBP decoding runs multiple BP decoders in parallel. Each of the decoders processes the vector of received symbols and uses a different parity-check matrix representation of the code. For this and all subsequent MBBP approaches we assume that the maximum number of iterations performed by each of the decoders is equal and given by $I_{\max}$. Let the parity-check matrix representation used by the ℓ-th decoder be denoted by $\boldsymbol{H}_\ell$, $\ell = 1, \ldots, l_{\mathrm{MBBP}}$. The corresponding decoded vector is referred to as $\hat{\boldsymbol{x}}_\ell$, $\ell = 1, \ldots, l_{\mathrm{MBBP}}$. In this context, l_{MBBP} is the number of decoder representations used in an MBBP setup. Each BP decoder reports whether it has found a valid codeword or not. We consider only the decoders that converged to a valid codeword as candidates for the final codeword and unite their indices to the set $\mathcal{V} \subseteq \{1, \ldots, l_{\mathrm{MBBP}}\}$. If no decoder finds a valid codeword, an arbitrary choice from the estimated words is performed, i.e. we let $\mathcal{V} = \{1, \ldots, l_{\mathrm{MBBP}}\}$. The optimal way to decide for $\hat{\boldsymbol{x}}$ from $\hat{\boldsymbol{x}}_v$, $v \in \mathcal{V}$, is to apply a least-metric selector (LMS) which determines the "best" representative using the decision rule $\hat{\boldsymbol{x}} = \underset{v \in \mathcal{V}}{\operatorname{argmax}} \Pr\left\{[Y_1, \ldots, Y_n] = [y_1, \ldots, y_n] \mid [\hat{X}_1, \ldots, \hat{X}_n] = [\hat{x}_{v,1}, \ldots, \hat{x}_{v,n}]\right\}$. For the AWGN channel it is well known that this equation can be rewritten as $\hat{\boldsymbol{x}} = \underset{v \in \mathcal{V}}{\operatorname{argmin}} \sum_{i=1}^{n} |y_\nu - \hat{x}_{v,i}|^2$. Note that the application of the operator $|\cdot|$ to a scalar number calculates the absolute value of this number. In other words, a full search is performed over a small set of candidates preselected by BP decoders. The estimated information vector $\hat{\boldsymbol{u}}$ can be obtained from $\hat{\boldsymbol{x}}$ in a straightforward manner if systematic encoding is applied or by a matrix multiplication in the case of non-systematic encoding.

Algorithm 7.1 (see page 102) gives an overview of the operations required to implement MBBP-NX-S decoding. All untagged commands and the ones tagged by MBBP-NX-S are required to implement the currently discussed approach. Commands tagged differently are needed to implement another MBBP-NX approach, to be introduced shortly. Figure 7.1 visualizes the MBBP-NX-S approach. It shows the parallel BP decoding units as well as the selector which forwards only one decoding candidate.

7.2.2 Parity-Check Matrices for MBBP Decoding of Algebraic Codes

A crucial factor for the success of the MBBP algorithm are the matrices $\boldsymbol{H}_\ell$, $\ell = 1, \ldots, l_{\mathrm{MBBP}}$, which the belief-propagation decoders run on. The better the performance of the single decoders

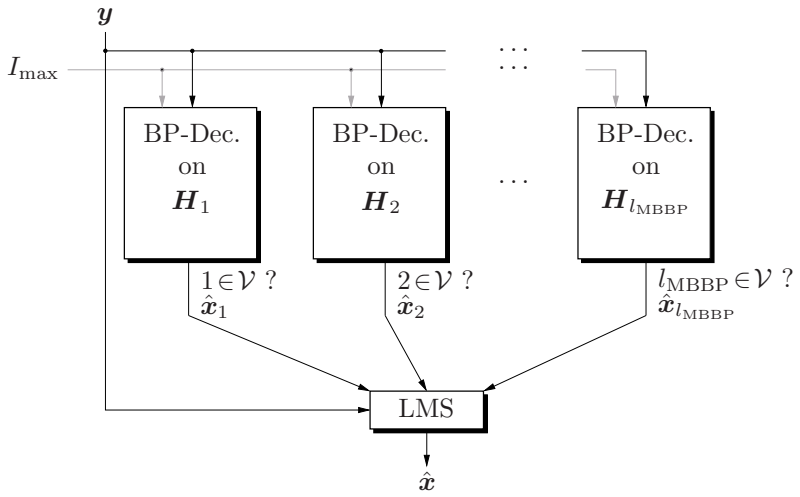


Figure 7.1: MBBP-NX-S decoding.

running on these matrices, the better the overall performance of the MBBP decoder. Clearly, adding further matrices with a poor BP performance to an existing MBBP setup cannot decrease the performance of the MBBP-NX-S approach. However, such an additional representation can increase the performance only by a marginal amount. Consequently, we need to find a set of matrices with good BP decoding performance in order to obtain good decoding results with a moderate number of decoder representations.

We discuss three kinds of codes in our investigations on the MBBP approach. In this section, we focus on cyclic and extended cyclic algebraic codes with a rich structure. Considerations on these codes are interesting for joint decoder implementations in the field of concatenated channel coding, where usually inner LDPC codes and outer algebraic codes are used. Results on these codes can also be understood as a proof of concept of this scheme. In Section 7.7 we consider LDPC codes. We show how the well-performing codes constructed by the PEG algorithm are used in an MBBP setup. Here, we discuss PEG codes of a length up to $n = 1000$. Further, we investigate performance improvements for a selection of codes of rate 1/2 from the IEEE 802.16e [IEE05] WiMAX standard.

The cyclic algebraic codes investigated in this thesis contain Hamming codes [MS77, Ch. 1], BCH codes [MS77, Ch. 9], quadratic-residue (QR) codes [MS77, Ch. 16], and the (extended) Golay code [MS77, Ch. 20]. When using cyclic codes, we only consider parity-check matrices of cyclic form, cf. Section 5.1.1. It was observed for the BEC and by means of further simulations for the AWGN channel that a cyclic representation offers similar or better performance when compared to a non-cyclic representation. Appendix C discusses this topic for the BEC in detail. However, the performance is not equal for all cyclic representations. A good starting point to find well-performing parity-check matrices of cyclic form is to identify codewords of minimum weight of the dual code and use them to identify COGs. It is worth pointing out that there exist n' non-equal cyclic shifts of a COG, where n' is either n or an integer divisor of n, depending on the COG. Further, using these n' cyclic shifts as rows of a matrix does not necessarily result

in a parity-check matrix of full rank. For this reason, we restrict our attention to COGs which allow to generate parity-check matrices of cyclic form and rank $n-k$. A necessary condition for this is given by $n' \geq n-k$. In the following, we confine the set of considered COGs even further.

For the class of cyclic algebraic codes, we observe that the parity-check matrices exhibit a special property. Although $n-k$ shifts of each COG are sufficient to describe the code, it turned out that matrices with n cyclic shifts are more amenable for BP decoding of this class of codes. Reasons for this include the fact that these codes are not designed for BP decoding. This holds also for matrices using only minimum-weight codewords of the dual code. The use of n cyclic shifts provides a more uniform error protection for the symbols in the codeword. We provide an intuitive explanation for this behavior in form of an example.

Example 7.2.1 *Given is a parity-check matrix of cyclic form, denoted by $\boldsymbol{H}$, as well as the first three rows of this matrix, denoted as a matrix by $\boldsymbol{H}_1$, cf. Equation (7.1).*

$$\boldsymbol{H} = \left[\begin{array}{cc|cccc|c} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \qquad \boldsymbol{H}_1 = \left[\begin{array}{cc|cccc|c} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \tag{7.1}$$

Let us first discuss $\boldsymbol{H}_1$. The seven variable nodes do not participate in the same number of parity-check equations, i.e. $\boldsymbol{H}_1$ has irregular column degree. There exist at least two variable nodes (including the first and last) which participate in exclusively one parity check, and therefore have very low probability of being correctly decoded. In this example, three variable nodes do only participate in one parity check. Considering $\boldsymbol{H}$, every variable node takes part in exactly four parity checks. As expected, simulation results indicate a better decoding performance for this setting.

Allow us to elaborate briefly on the result of Example 7.2.1. If a variable node takes part in only one parity check, the concept of the iterative BP decoding algorithm is pointless. A variable node which is connected to only one parity-check node sends its intrinsic information p_i^{int} to the check node in every iteration. Such a variable node does not contribute to the evolvement of the iterative process at all. Being connected to one check node only, this variable node is prone to errors. Designed LDPC codes usually have a minimum left degree greater 1 [Urb].

In order to generate redundant parity-check matrices of cyclic form with n non-equal rows, we restrict the choice of n' further and postulate $n' = n$. Besides this postulation, we further assume for each considered COG that it allows to create a parity-check matrix of cyclic form and full rank. In a nutshell, we henceforth assume that a parity-check matrix consists of n cyclic shifts of the corresponding COG and has full rank.

Let us now investigate the selection of the COG in more detail. We show that an entity from the field of iterative BEC decoding serves well as an evaluation criterion to compare different bases with respect to their decoding performance. The number of stopping sets of size σ, $1 \leq \sigma \leq d$, is used to provide a first estimate on the BER and FER performance of the parity-check matrix. The number of unresolved stopping sets up to size d is a good criterion to compare the performance of different parity-check matrix representations. Reasons for this include the fact that pseudocodewords[1] observed on other channels than the BEC represent stopping sets in the Tanner graph [Fel03, KS07].

In order to find a possibly large set of well-performing matrices, we make use of the following facts. We observe that several cyclic parity-check matrices have similar BP decoding performance and that for these matrices, the performance criterion asserts equal decoding performance for the BEC. In order to group COGs with this property, we partition them into COG families. In this coherence we define $\mathcal{D}$ as the set of COGs.

Definition 7.2.1 *We define the "stopping set performance" as the number of stopping sets of size up to d. Strictly speaking, this definition only holds for the BEC, but it will also be used in connection with the AWGN channel. Partition the set of indices $\{1, \ldots, |\mathcal{D}|\}$ into F subsets $\mathcal{F}_f$, $f \in \{1, \ldots, F\}$, such that the stopping set performance of $\boldsymbol{H}_a$ generated from COG_a is equal for all $a \in \mathcal{F}_f$. We refer to the set $\{\mathrm{COG}_a\}$, $a \in \mathcal{F}_f$, as the f-th* COG family.

In other words, Definition 7.2.1 says that the sets

$$\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_F = \{1, \ldots, |\mathcal{D}|\}$$

with

$$\mathcal{F}_f \cap \mathcal{F}_{f'} = \emptyset,\ f \neq f',$$

are created such that the stopping set performance is constant for all $\boldsymbol{H}_a$ created from COG_a with $a \in \mathcal{F}_f$, $f \in \{1, \ldots, F\}$.

With rising co-dimension it becomes infeasible to determine all COGs of a given code by evaluating the performance or counting stopping sets. However, for cyclic algebraic codes, we propose a scheme to generate subsets of COG families. Examples show that often whole families are generated. This approach is very amenable for the MBBP decoder setup. Given one single COG, it allows to generate the corresponding family of COGs and from this, a set of cyclic parity-check matrices with equal stopping set performance and comparable decoding performance on the AWGN channel. This is accomplished by applying permutations from the automorphism group of the code to the given COG. The following definition is useful in our subsequent derivations.

[1] The set of pseudocodewords of a code is a superset of the actual codewords [Fel03].

Definition 7.2.2 *Let ζ be a permutation operating on a vector $\boldsymbol{x}_1$, resulting in $\boldsymbol{x}_2 = \zeta(\boldsymbol{x}_1)$. We define the ζ-permutation of an $m \times n$ matrix $\boldsymbol{M}_1$ to be a matrix obtained by permuting each row of $\boldsymbol{M}_1$ according to ζ. In this setting, $\boldsymbol{M}_2 = \zeta(\boldsymbol{M}_1)$ is used to denote $\boldsymbol{M}_2(j,:) = \zeta(\boldsymbol{M}_1(j,:))$, $j = 1, \ldots, m$.*

The results to be derived rely on the fact that $\gamma(\alpha^j(\boldsymbol{x})) = \alpha^{j'}(\gamma(\boldsymbol{x}))$, where $\alpha(\boldsymbol{x})$ is the cyclic-shift permutation (P_1), defined in Section 6.3.2 and $\gamma(\boldsymbol{x})$ is an affine permutation, defined below.

Lemma 7.2.3 *We are interested in a necessary and sufficient condition for*

$$\begin{aligned} \gamma(\alpha^j(\boldsymbol{x})) &= \alpha^{j'}(\gamma(\boldsymbol{x})), \\ &\quad \text{for all } j \in \{1, \ldots, n\}, \text{ and some } j' \in \{1, \ldots, n\}, \end{aligned}$$

to hold, such that there exists a one-to-one correspondence between j and j'. This is the case if $\gamma(\cdot)$ is an affine permutation, defined as

$$\begin{aligned} \gamma: \quad & i \to (q \cdot i + \theta_\gamma) \bmod^* n, \\ & \text{for some } q \in \{1, \ldots, n\},\ \theta_\gamma \in \{1, \ldots, n\}, \text{ and } i = 1, \ldots, n, \end{aligned}$$

with $\gcd(q, n) = 1$. Here, $\gcd(q, n)$ denotes the greatest common divisor (GCD) of q and n, and θ_γ is an offset in the function $\gamma(\boldsymbol{x})$.

Proof 7.2.4 *The claim of Lemma 7.2.3 can be rewritten as*

$$\begin{aligned} \gamma_i(i + j) &= \gamma_i(i) + j' \bmod^* n \\ &\quad j \in \{1, \ldots, n\},\ j' \in \{1, \ldots, n\},\ i = 1, \ldots, n, \end{aligned}$$

where $\gamma_{\mathrm{i}}(i)$ returns the index of $\gamma(x_i)$. The former equality is true iff $\gamma(i)$ is a linear function of the form $q \cdot i + \theta_\gamma$ and if $gcd(q, n) = 1$. Consequently, there exists a one-to-one correspondence between j' and j. ■

The lemma asserts that cyclic permutations commute (up to a cyclic shift) with all affine permutations in a symmetric group.

Example 7.2.2 *Consider $n = 5$, $j = 1$, and $\gamma: i \to (2i + 2) \bmod^* n$, $i = 1, \ldots, n$. We use the codeword 10100, where index 1 is located on the leftmost position. Then, $\gamma(\alpha(10100)) = \alpha^2(\gamma(10100)) = 10001$ holds. We observe that $j = 1$ and $j' = 2$ is true in this example.*

Let us now investigate the two permutations P_1 and P_2, cf. 6.3.2, which are common for all cyclic codes.

Claim 7.2.5 *If* $\mathrm{COG}_b = \alpha^j(\mathrm{COG}_a)$*, for some* $j \in \{1, \ldots, n\}$*, then*

$$\overline{\Sigma}_\sigma(\boldsymbol{H}_a) = \overline{\Sigma}_\sigma(\boldsymbol{H}_b), \text{ for all } \sigma \leq n.$$

Be reminded that there exists a one-to-one correspondence between $\boldsymbol{H}_a$ and the underlying COG_a. The preceeding claim asserts that the number of stopping sets of size up to n does not change by applying the permutation α^j. This result includes the property that $\boldsymbol{H}_a$ and $\boldsymbol{H}_b$ have equal stopping set performance.

Proof 7.2.6 *Applying* α^j *to* COG_a *results in cyclically permuting the rows in* $\boldsymbol{H}_a$*. This cyclic permutation preserves all stopping sets. This concludes the proof.* ∎

Claim 7.2.7 *If* $\mathrm{COG}_c = \beta^j(\mathrm{COG}_a)$*, for some* $j \in \{0, \ldots, h-1\}$*, then*

$$\overline{\Sigma}_\sigma(\boldsymbol{H}_a) = \overline{\Sigma}_\sigma(\boldsymbol{H}_c), \text{ for all } \sigma \leq n.$$

Be reminded that the permutation β *was defined in Section 6.3.2.*

Proof 7.2.8 *Since*

$$\begin{aligned} \boldsymbol{H}_c(j,:) &= \alpha^{j-1}(\boldsymbol{H}_c(1,:)) = \alpha^{j-1}(\beta^j(\boldsymbol{H}_a(1,:))) \\ &= \beta^j(\alpha^{j'-1}(\boldsymbol{H}_a(1,:))), \text{ for all } j \in \{1,\ldots,n\} \text{ and some } j' \in \{1,\ldots,n\}. \end{aligned}$$

which follows from the fact that β^j *is an affine permutation with* $q = 2$ *and* $\theta_\gamma = n$*. As a result, the transformation from* $\boldsymbol{H}_a$ *to* $\boldsymbol{H}_c$ *can be described by simple row- and column-permutations. Note that the column permutations are applied to the matrix, what differs in general from the approach of permuting a given* COG *and using the result to create a new parity-check matrix. However, for the special choice of permutations in connection with circulant matrices, these approaches are equivalent. The intersection numbers of any pair of rows do not change by the permutation described in Claim 7.2.7. Although the set of unresolved stopping sets changes, its magnitude stays constant.* ∎

We conclude that affine permutations in the automorphism group $\mathrm{Aut}(\mathcal{C})$ of a code allow to map a COG of a given family to another COG of this family. This is a welcome property if one intends to generate an MBBP setup for a given code. Instead of storing a set of matrices only a small set of COGs is required. For many decoding scenarios even one COG may be sufficient. After generating the required set of COGs, one can create the matrices required for the MBBP setup by cyclically shifting the corresponding COGs.

7.3 Case Study: A BCH Code in an MBBP-NX-S Setup

Before diving deeper into the field of MBBP decoding, allow us to demonstrate its benefits using an algebraic code of small dimension and co-dimension. We choose the $[31, 16, 7]$ BCH code for

demonstration purposes. Its dual code contains codewords of minimum weight 8. Further, three COGs are sufficient to generate all 15 COGs by means of the permutation β. All the COGs lead to an equal stopping set performance such that $F = 1$ holds for this code.

We briefly demonstrate three advantages of the MBBP-NX-S approach when compared to standard BP. First, when allowing for higher complexity, a significant performance improvement compared to standard BP is obtained. Second, when limiting the number of overall iterations performed in an MBBP-NX-S setup and varying l_{MBBP}, it is to observe that the choice $l_{\mathrm{MBBP}} > 1$ leads to better results than standard BP decoding. Third, the MBBP-NX-S decoder is able to trade iterations with additional parallel representations, allowing the user to decide between a short decoding delay or a small number of parallel realizations.

We present a small selection of simulation results to confirm the propositions above. Up to 15 BP decoders run in parallel to decode the $[31, 16, 7]$ BCH code after signaling over AWGN. Two scenarios are investigated. First, *each decoder* is allowed to perform up to $I_{\max} \leq 100$ iterations to decode one frame. Second, a constraint on the total number of iterations is given. Here, the decoders are allowed to perform not more than 100 and 200 iterations in total, respectively. Once this number is set, the iterations are equally shared to the decoding units. This allows to compare the decoding performance under comparable complexity settings to a single decoding unit. Figure 7.2 shows the SNR as $10 \cdot \log_{10}(E_{\mathrm{b}}/N_0)$ which is required to obtain FER $= 10^{-3}$ when signaling over the AWGN channel.

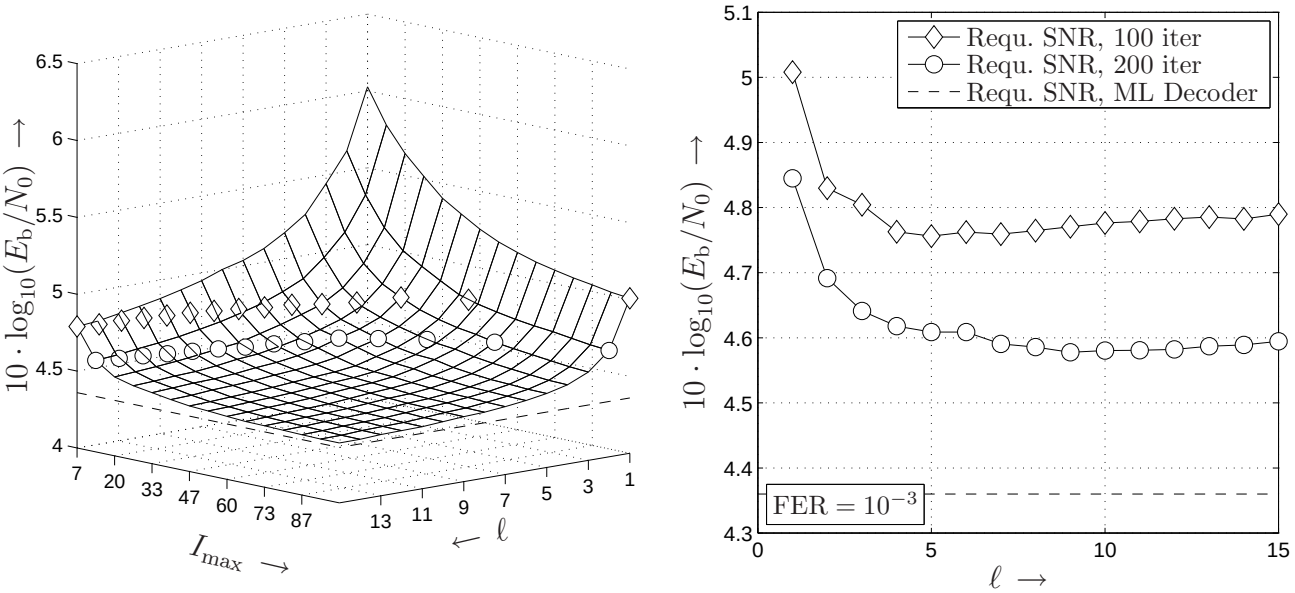


Figure 7.2: Required SNR ($10 \cdot \log_{10}(E_{\mathrm{b}}/N_0)$) to obtain FER $= 10^{-3}$.

On the left-hand side of Figure 7.2 it is to observe that a significant performance improvement can be obtained by using MBBP-NX-S. Note that the dashed lines in this figure mark the SNR required by the ML decoder. Diamonds and circles mark the SNR required for FER $= 10^{-3}$ when the constraint $I_{\max} \cdot l_{\mathrm{MBBP}} = 100$ and $I_{\max} \cdot l_{\mathrm{MBBP}} = 100$, respectively, is set. The right-hand

side of Figure 7.2 shows a two-dimensional projection of the marked points. We observe that the performance of an ML decoder cannot be obtained with a total of 200 iterations. However, it can be seen that if 100 iterations are allowed, they should be shared on 5 decoders instead of 1 decoder. If 200 iterations are available, 9 decoders are a favorable choice. Concluding, we would like to mention that 15 parallel decoders are a realizable setup considering the current achievements in microelectronics, where CPUs with 80 parallel cores are being realized [VHR+07]. If this computational power is not available, MBBP can be used as a post-processing step for standard BP, decoding frames for which standard BP did not converge.

A case study of this type was also conducted for the $[24, 12, 8]$ extended Golay code, and can be found in [HHLM07].

7.4 Further Types of MBBP Decoding

Having understood that MBBP decoding is a valuable tool to improve the decoding performance of iterative decoders, we investigate different variations of this approach. In general, we distinguish between two kinds of MBBP decoding. We refer to the first class of MBBP decoding as *MBBP-NX* decoding. Be reminded that the qualifier NX states that no information is exchanged. Besides the standard MBBP approach without information exchange, i.e. the MBBP-NX-S approach, the MBBP-NX algorithms class contains an approach that stops decoding as soon as a first decoder converges to a codeword. We refer to this scheme as First-Success MBBP-NX (*MBBP-NX-FS*) decoding. This decoder has lower complexity and delay when compared to the standard MBBP-NX-S and approaches its performance closely. A second category of MBBP decoders are MBBP-X decoders, which realize the promising idea of decoder communication during the decoding process, i.e. synchronizing a subset of their messages after a given number of iterations. However, it turns out that MBBP-X decoders are not too amenable for decoding of dense cyclic codes. Reasons for this include the fact that information exchange generates new cycles which stretch across different decoders. Nevertheless, decoders of this type often outperform a standard BP setup what motivates their discussion.

7.4.1 MBBP-NX Decoding

The most important MBBP-NX Decoder, the MBBP-NX-S approach, has already been discussed above. In order to complete the set of MBBP-NX decoders, we investigate a decoder that follows the approach of MBBP-NX-S closely but allows for a lower number of iterations.

First-Success MBBP Decoding (MBBP-NX-FS)

The central element of this algorithm is a selection unit which monitors all decoders and stops the decoding process as soon as one of the decoders has found a valid codeword. We denote the number of iterations required by the ℓ-th decoder to converge by I_ℓ. The least-iteration

selector (LIS) forwards the estimated codeword from the decoder that converged first, indexed by ℓ^*. Subsequently, it terminates the whole decoding process. Figure 7.3 presents the approach and shows in particular the path of the termination signal which runs from the selection unit to the BP decoders. The estimate $\hat{\boldsymbol{x}}$ obtained by the decoder indexed by ℓ^* is forwarded to the information sink. In order to consider the case where at least two decoders find a valid codeword simultaneously, we choose the forwarded codeword randomly from $\boldsymbol{x}_v$, $v \in \mathcal{V}$, where $\mathcal{V}$ is defined as for the MBBP-NX-S decoder. Note that this approach requires only $I_{\ell^*} = \min_\ell I_\ell$ iterations to converge. The MBBP-NX-FS algorithm cannot outperform its MBBP-NX-S counterpart in terms of decoding performance. However, it provides for significant time savings, which make it amenable for use in low-complexity systems, which require a better decoding performance than BP can provide.

The steps of these algorithms are summarized in Algorithm 7.1. All untagged commands are executed both during the MBBP-NX-S and the MBBP-NX-FS procedure, whereas tagged commands are only executed within the algorithm indicated by the tag.

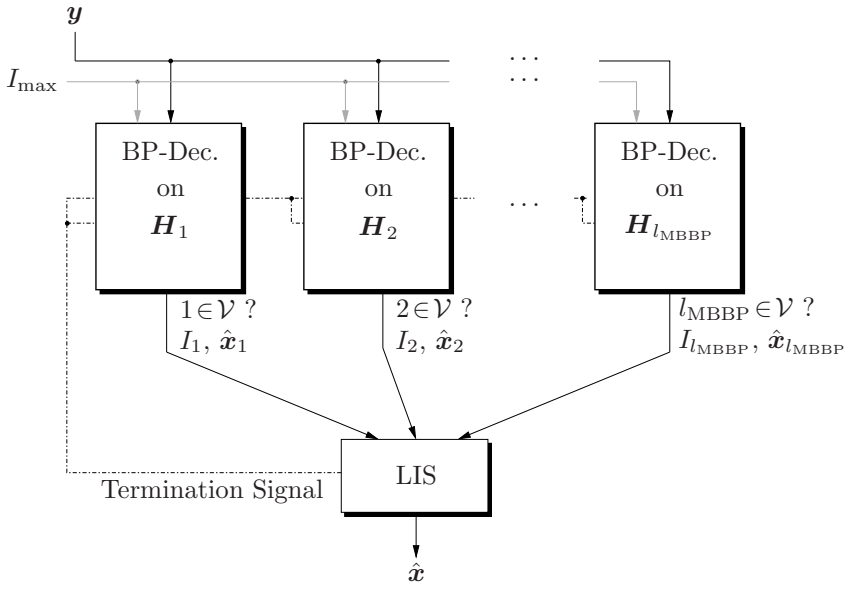


Figure 7.3: MBBP-NX-FS decoding.

7.4.2 MBBP-X Decoding

We focus on MBBP approaches which *periodically exchange* information *during* the decoding process. This means that the decoders are self-acting most of the time, but pause the decoding process in order to communicate with each other after a given number of iterations, $I = a \cdot I_{\mathrm{sync,MBBP}}$, $a \in \mathbb{N}$, $I \leq I_{\max}$. In this step, extrinsic information on specific sets of variable nodes is exchanged. Note that the *intrinsic information* of a given variable node depends only on the channel output $\boldsymbol{y}$ and is therefore equal for all decoders and all iterations. Figure 7.4 depicts the basic architecture of an MBBP-X decoder.

Algorithm 7.1 MBBP-NX: Standard (NX-S) and First-Success (NX-FS).

Input: $\boldsymbol{y}$, l_{MBBP}, $\boldsymbol{H}_\ell$ with $\ell = 1, \ldots, l_{\mathrm{MBBP}}$, $I_{\max}$
Output: $\hat{\boldsymbol{x}}$
Function call: $\hat{\boldsymbol{x}}$=mbbp_nx($\boldsymbol{y}$, l_{MBBP}, $\boldsymbol{H}_\ell$ with $\ell = 1, \ldots, l_{\mathrm{MBBP}}$, $I_{\max}$)

1: $\mathcal{V} = \emptyset$, $I = 1$
2: **while** $I \leq I_{\max}$ (NX-S, NX-FS) **and** $\mathcal{V} == \emptyset$ (NX-FS) **do**
3: $[\hat{\boldsymbol{x}}_\ell, \text{success\_flag}_\ell, p_{ij}^{\text{ext}}(\boldsymbol{H}_\ell)] = \text{bp\_decode}(\boldsymbol{y}, \boldsymbol{H}_\ell, 1, p_{ij}^{\text{ext}}(\boldsymbol{H}_\ell))$, $i = 1, \ldots, n$, $j \in \mathcal{N}(\mathrm{v}_i)$, $\ell \in \{1, \ldots, l_{\mathrm{MBBP}}\} \backslash \mathcal{V}$
4: **for** $\ell = 1, \ldots, l_{\mathrm{MBBP}}$ **do**
5: **if** $\text{success\_flag}_\ell == 1$ **then**
6: $\mathcal{V} = \mathcal{V} \cup \ell$
7: **end if**
8: **end for**
9: $I = I + 1$
10: **end while**
11: **if** $\mathcal{V} == \emptyset$ **then**
12: $\mathcal{V} = \{1, \ldots, l_{\mathrm{MBBP}}\}$
13: **end if**
14: $\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}_v,\, v \in \mathcal{V}}{\operatorname{argmin}} \sum_{i=1}^{n} |y_i - \operatorname{map}(x_{v,i})|^2$ (MBBP-NX-S)
15: $\hat{\boldsymbol{x}} = \boldsymbol{x}_{v^*}$ with $v^* = \underset{v \in \mathcal{V}}{\operatorname{argmin}} I_v$ (MBBP-NX-FS)

The messages exchanged between decoders represent extrinsic information. As the graph structure between the different representations is not equal, we collect and combine all available information at each variable node before the exchange is conducted. Hence, the information sent out from each decoder and for each variable node v_i, $i = 1, \ldots, n$ is denoted by $p_i^{\text{ext}}(\boldsymbol{H}_\ell)$, where $p_i^{\text{ext}}(\boldsymbol{H}_\ell) = \boxtimes_{j \in \mathcal{N}(\mathrm{v}_i)} p_{ij}^{\text{ext}}(\boldsymbol{H}_\ell)$. The argument $\boldsymbol{H}_\ell$ specifies the representation these probabilities emanate from. Subsequently, this information is processed by one of the three methods described below: probability-averaging, selection according to the highest reliability, and normalized multiplication. The processed probability cannot be attributed to a particular matrix representation and is hence denoted as p_i^{ext}. Let us now describe in detail how the decoders cooperate when data is exchanged. First, all decoders report their estimated extrinsic probability for each variable node. A function selects a subset of decoders for each variable node and processes their outputs to generate one estimate of that probability using one of the three methods introduced. Using the established extrinsic information and the intrinsic information from the channel, each decoder calculates the a-posteriori probabilities corresponding to the variable nodes and resumes its own decoding process. The estimate $\hat{\boldsymbol{x}}$ is selected by the communicating-decoders selector (CDS). This device is active if $I = a \cdot I_{\text{sync,MBBP}}$, $a \in \mathbb{N}$ and checks if at least one decoder has converged. If so, it selects one of these decoders randomly.

In the following, we present three different variations of MBBP-X decoding which differ in the function processing the extrinsic information from the decoders.

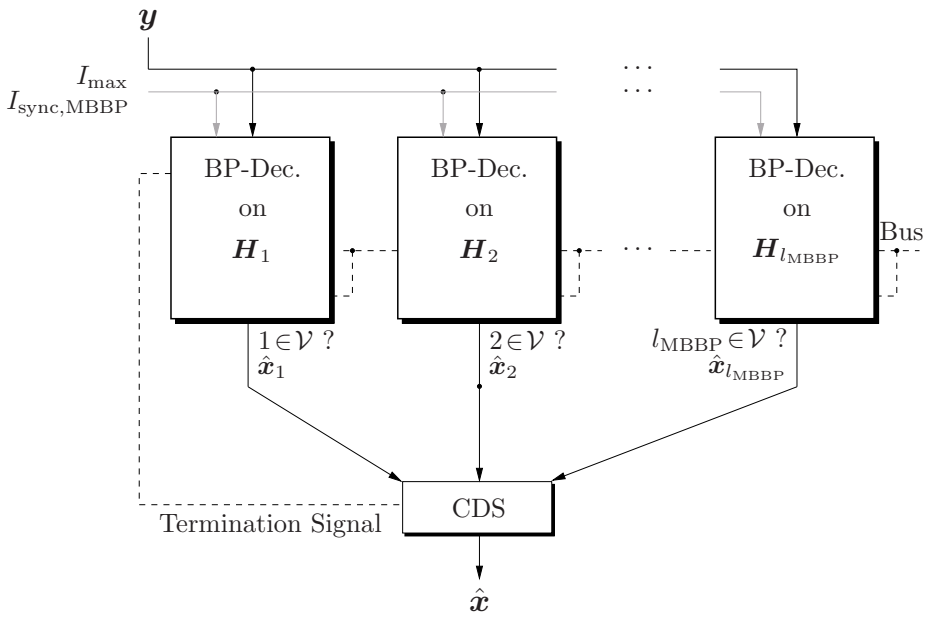


Figure 7.4: MBBP-X decoder architecture. The bus is active at iteration $I = a \cdot I_{\text{sync,MBBP}}$, $a \in \mathbb{N}$, $I \leq I_{\max}$.

Probability-Averaging MBBP-X (MBBP-X-PA)

A first MBBP-X approach performs *averaging* to determine the extrinsic probabilities used in the subsequent course of the decoding process. Probabilistic averaging was used in [LM05] in order to lower the error-floor with standard BP decoding. We use *arithmetic averaging* to find an agreement on the extrinsic probability sent out from a variable node. Note that this calculation is done separately for each variable node. Prior to the processing step itself, a soft-metric majority-vote $\bar{v}_i$ is calculated for variable node i, $i = 1, \ldots, n$, as

$$\bar{v}_i = \sum_{\ell=1}^{l_{\text{MBBP}}} \ln\left(\frac{p_i^{\text{ext}}(\boldsymbol{H}_\ell)}{1 - p_i^{\text{ext}}(\boldsymbol{H}_\ell)}\right). \tag{7.2}$$

Only the decoders which concur with $\bar{v}_i$ are taken into account in the averaging process for calculating p_i^{ext}. The subset of decoders used in the described averaging process is defined as

$$\mathcal{A}_i = \left\{\ell | \text{sgn}(\bar{v}_i) = \text{sgn}\left(\ln\left(\frac{p_i^{\text{ext}}(\boldsymbol{H}_\ell)}{1 - p_i^{\text{ext}}(\boldsymbol{H}_\ell)}\right)\right)\right\}, \; i = 1, \ldots, n. \tag{7.3}$$

The function $\text{sgn}(\cdot)$ determines the sign of a variable. Note that the calculation of $\bar{v}_i$ is equivalent to the normalized multiplication expressed by log-likelihood ratios, cf. Equation (3.15). However, only the sign of $\bar{v}_i$ is used for further processing, see Equation (7.3). We use only decoders with an index in $\mathcal{A}_i$, as the decoders tend to have strongly different opinions for a given variable node. Simulations with and without this feature confirm its positive effect on the decoding result.

The processing step itself is conducted by

$$p_i^{\text{ext}} = \frac{1}{|\mathcal{A}_i|} \sum_{\ell' \in \mathcal{A}_i} p_i^{\text{ext}}(\boldsymbol{H}_{\ell'}),\ i = 1, \ldots, n. \tag{7.4}$$

We point out that for implementation of this algorithm the proposed steps need to be followed exactly in order to reproduce the results presented in this thesis. If counter probabilities are required, these values should be generated by subtracting p_i^{ext} from 1.

Highest-Reliability MBBP-X (MBBP-X-HR)

This approach assumes that variable nodes with a high reliability, i.e. a probability close to zero or one, are more likely to be correct than others [VF04]. The method to be introduced also deploys the weighted majority-vote introduced in Equation (7.2) to select the decoders used for computing the information update for a given variable node.

In this approach, the processing unit selects the most reliable information on a variable node from all considered decoders. This operation reads

$$p_i^{\text{ext}} = p_i^{\text{ext}}(\boldsymbol{H}_{\ell^*}) \tag{7.5}$$

with $\ell^* = \underset{\ell' \in \mathcal{A}_i}{\operatorname{argmax}} \ |p_i^{\text{ext}}(\boldsymbol{H}_{\ell'}) - 0.5|$, $i = 1, \ldots, n$, and where $\mathcal{A}_i$ is given according to Equation (7.3).

Normalized-Multiplication MBBP-X (MBBP-X-NM)

In Section 3.1 it was shown that the *normalized multiplication* is the optimal way to determine a probability for a given event if at least two independently distorted observations are available. In this paragraph, we discuss the use of the normalized multiplication formula for the processing step of the MBBP-X setup.

The extrinsic information provided by the parallel decoders depends on the underlying parity-check matrices, but also on the received codeword $\boldsymbol{y}$. One can understand the extrinsic information for a variable node obtained from different decoders as multiple observations. Recall that these representations are yet strongly dependent. Inspired by the belief-propagation algorithm, which uses normalized multiplication although the data is not independent, we propose an MBBP-X algorithm which uses normalized multiplication for the processing step. Again, only decoders which agree with the soft-metric majority-vote according to Equation (7.2) are taken into account. The "combined" probability distribution of the binary variable X_i is given by

$$p_i^{\text{ext}} = \frac{\prod\limits_{\ell' \in \mathcal{A}_i} p_i^{\text{ext}}(\boldsymbol{H}_{\ell'})}{\prod\limits_{\ell' \in \mathcal{A}_i} p_i^{\text{ext}}(\boldsymbol{H}_{\ell'}) + \prod\limits_{\ell' \in \mathcal{A}_i} (1 - p_i^{\text{ext}}(\boldsymbol{H}_{\ell'}))},\ i = 1, \ldots, n. \tag{7.6}$$

A summary of the steps for the three algorithms from the class of MBBP-X decoders is given in Algorithm 7.2. As before, the untagged commands are executed in all three algorithms, while tags specify in which MBBP-NX variation the current command is to be executed.

Algorithm 7.2 MBBP-X: Probability-Averaging (PA), Highest-Reliability (HR), and Normalized-Multiplication (NM).

Input: $\boldsymbol{y}$, l_{MBBP}, $\boldsymbol{H}_\ell$ with $\ell = 1, \ldots, l_{\text{MBBP}}$, $I_{\max}$, $I_{\text{sync,MBBP}}$
Output: $\hat{\boldsymbol{x}}$
Function call: $\hat{\boldsymbol{x}}$=mbbp_x($\boldsymbol{y}$, l_{MBBP}, $\boldsymbol{H}_\ell$ with $\ell = 1, \ldots, l_{\text{MBBP}}$, $I_{\max}$, $I_{\text{sync,MBBP}}$)

$\mathcal{V} = \emptyset$, $I = 1$
$p_i^{\text{ext}}(\boldsymbol{H}_\ell) = 0.5$, $i = 1, \ldots, n$, $\ell = 1, \ldots, l_{\text{MBBP}}$
while $I \leq I_{\max}$ **and** $\mathcal{V} == \emptyset$ **do**
  $[\hat{\boldsymbol{x}}_\ell, \text{success\_flag}_\ell, p_{ij}^{\text{ext}}(\boldsymbol{H}_\ell)] = \text{bp\_decode}(\boldsymbol{y}, \boldsymbol{H}_\ell, 1, p_i^{\text{ext}}(\boldsymbol{H}_\ell))$, $i = 1, \ldots, n$, $j \in \mathcal{N}(\text{v}_i)$, $\ell = 1, \ldots, l_{\text{MBBP}}$
  $p_i^{\text{ext}}(\boldsymbol{H}_\ell) = \boxtimes_{j \in \mathcal{N}(\text{v}_i)} p_{ij}^{\text{ext}}(\boldsymbol{H}_\ell)$
  if $I == a \cdot I_{\text{sync,MBBP}}$, $a \in \mathbb{N}$ **then**
    Perform Eq. (7.2) $\rightarrow \bar{v}_i$
    Perform Eq. (7.3) $\rightarrow \mathcal{A}_i$
    Perform Eq. (7.4) $\rightarrow p_i^{\text{ext}}$, $i = 1, \ldots, n$ (MBBP-X-PA)
    Perform Eq. (7.5) $\rightarrow p_i^{\text{ext}}$, $i = 1, \ldots, n$ (MBBP-X-HR)
    Perform Eq. (7.6) $\rightarrow p_i^{\text{ext}}$, $i = 1, \ldots, n$ (MBBP-X-NM)
    for $\ell = 1, \ldots, l_{\text{MBBP}}$ **do**
      $p_i^{\text{ext}}(\boldsymbol{H}_\ell) = p_i^{\text{ext}}$
      if $\text{success\_flag}_\ell == 1$ **then**
        $\mathcal{V} = \mathcal{V} \cup \ell$
      end if
    end for
  end if
  $I = I + 1$
end while
if $\mathcal{V} == \emptyset$ **then**
  $\mathcal{V} = \{1, \ldots, l_{\text{MBBP}}\}$
end if
$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}_v,\, v \in \mathcal{V}}{\text{randfun}}$

7.5 Performance Results for MBBP Decoding

We present simulation results for the $[24, 12, 8]$ extended Golay code and the $[127, 64, 21]$ BCH code, which serve as exemplary representatives for various algebraic codes. For these codes and other algebraic codes tested, MBBP-NX approaches perform significantly better than MBBP-X approaches.

	$\left\lvert\overline{\Sigma}_\sigma\left(\boldsymbol{H}_a\right)\right\rvert$, $a \in$		
	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$
$\sigma \leq 5$	0	0	0
$\sigma = 6$	0	437	46
$\sigma = 7$	1357	10143	1495
$\sigma = 8$	25783	73209	20631

Table 7.1: Number of stopping sets in the $[24, 12, 8]$ extended Golay code when using parity-check matrices $\boldsymbol{H}_a$, $\mathrm{COG}_a \in \mathcal{F}_f$, $f = 1, \ldots, 3$.

7.5.1 Simulation Results for the $[24, 12, 8]$ Extended Golay Code

We show results for the $[24, 12, 8]$ extended Golay code and compare the performance when using all possible COGs. At this point, we are especially interested in the maximum performance obtainable by the novel approaches.

Exemplary for the Golay code, we describe the construction process of the matrices used in an MBBP setup in detail. This code is an extended cyclic code and we assume that the overall parity bit is on the last position, cf. Section 6.3.2. We construct the parity-check matrices from each COG in terms of 23 shifts performed on the first 23 positions. The overall parity bit on position 24 is not included in the cyclic shift but stays put. For the 24-th row of the parity-check matrix, we use the all-one codeword. We point out that there are choices for the 24-th parity-check equation that lead to better decoding performance. However, this codeword preserves the stopping set distribution of the 23×24 matrix, and is invariant under all affine permutations. This includes the permutations P_1 and P_2.

We consider one representative COG for each family, termed $\mathrm{COG}_{\mathcal{F}_f}$, $f = 1, 2, 3$, listed in binary notation below,

$$\begin{aligned}
\mathrm{COG}_{\mathcal{F}_1} &= [1\,1\,0\,1\,0\,1\,0\,0\,1\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0], \\
\mathrm{COG}_{\mathcal{F}_2} &= [1\,1\,1\,0\,0\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1], \\
\mathrm{COG}_{\mathcal{F}_3} &= [1\,1\,0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,0].
\end{aligned}$$

Each $\mathrm{COG}_{\mathcal{F}_f}$, $f \in \{1, 2, 3\}$ allows for generating all COGs in the family $\mathcal{F}_f$ by repeated application of permutation P_2. Note that in general the set of generated COGs is only a subset of all COGs in the corresponding family. For this code, 33 COGs exist and each $\mathrm{COG}_{\mathcal{F}_f}$, $f = 1, 2, 3$, allows for generating 11 different representations. The permutation P_2 is an automorphism of an extended cyclic code and preserves the stopping set distribution of the matrices $\boldsymbol{H}_\ell$, $\ell \in \mathcal{F}_f$, if the check in row 24 is invariant under P_2. The numbers of unresolved stopping sets in $\boldsymbol{H}_\ell$, $\ell \in \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3\}$, are listed in Table 7.1.

Figure 7.5 shows the BER and FER performance of MBBP-NX-S decoding and MBBP-NX-FS decoding of the extended Golay code. Each family is shown separately.

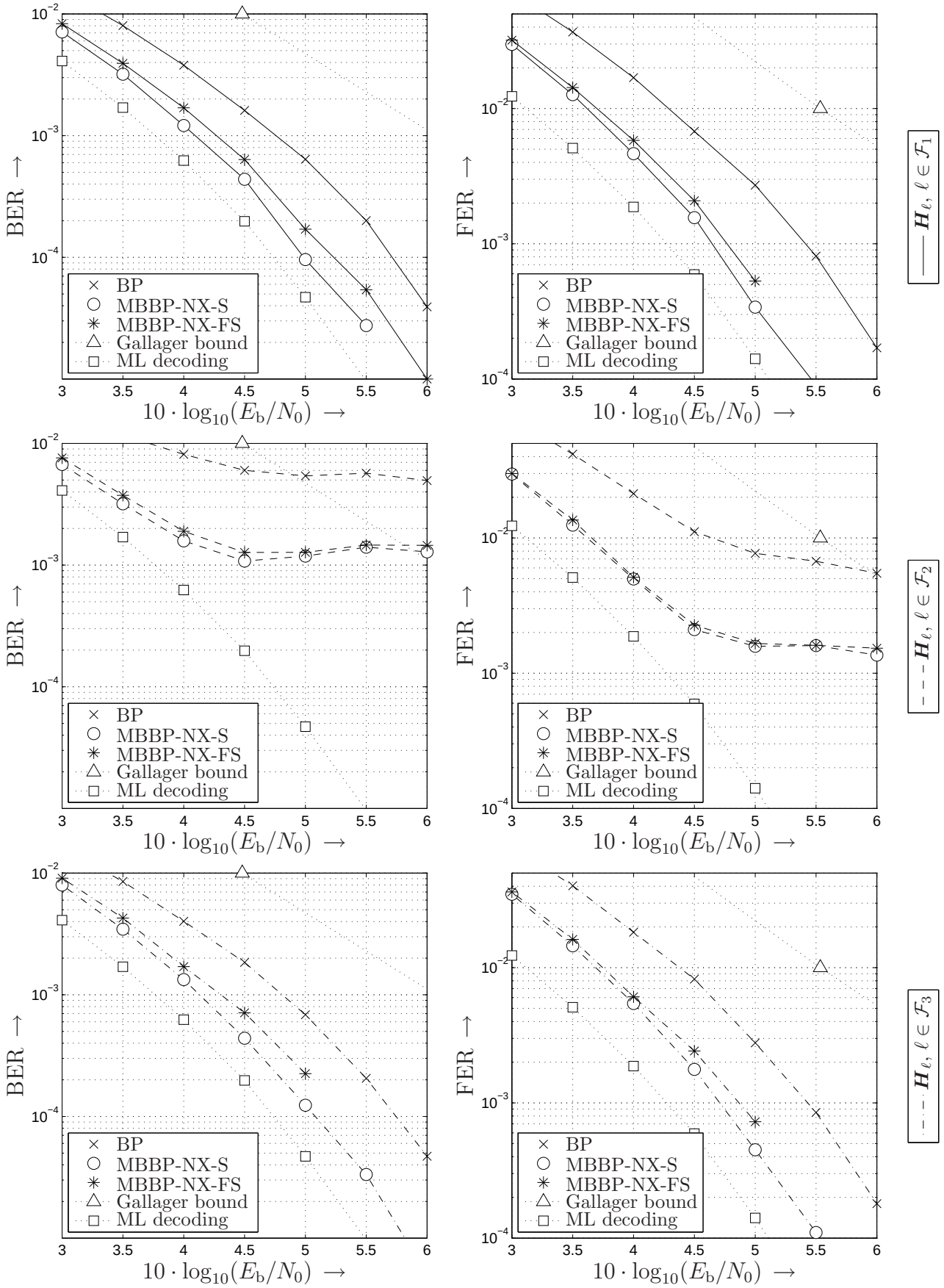


Figure 7.5: Performance comparison for the $[24, 12, 8]$ extended Golay code using $\boldsymbol{H}_\ell$, $\ell \in \mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$. The BP decoder and MBBP-NX decoders with $l_{\mathrm{MBBP}} = 11$ and $I_{\max} = 100$ are applied.

Figure 7.6 depicts the performance of the MBBP-X-PA, MBBP-X-HR, and MBBP-X-NM approach exemplary for the families $\mathcal{F}_1$ and $\mathcal{F}_2$. For all decoding types, $l_{\mathrm{MBBP}} = 11$ parallel BP decoders from each family are used. We set $I_{\mathrm{sync,MBBP}} = 10$ and $I_{\max} = 100$ for the simulations, but report that many other reasonable settings for $I_{\mathrm{sync,MBBP}}$ lead to similar results.

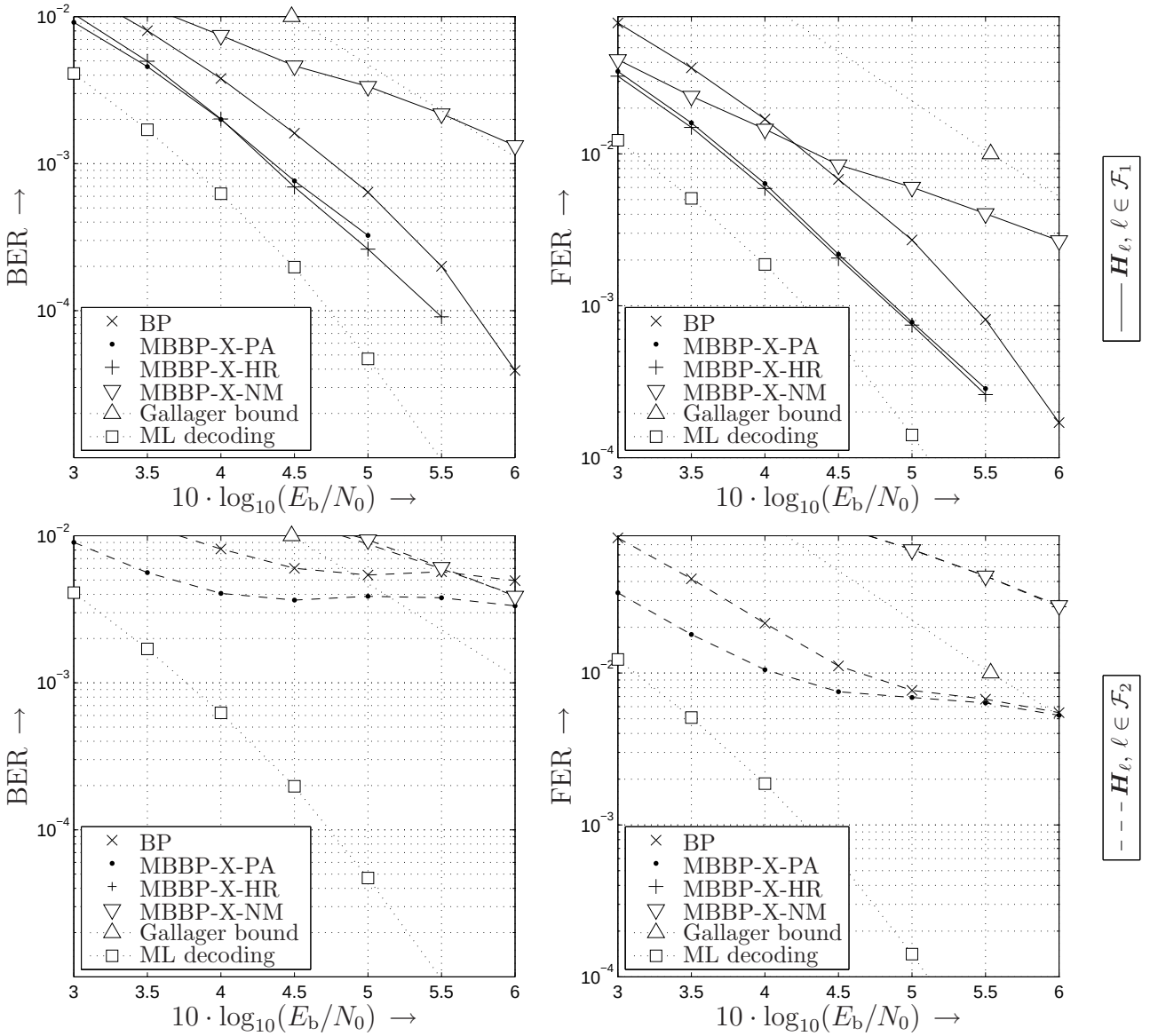


Figure 7.6: Performance comparison for the $[24, 12, 8]$ extended Golay code using $\boldsymbol{H}_\ell$, $\ell \in \mathcal{F}_1$, $\mathcal{F}_2$. The BP decoder and MBBP-X decoders with $l_{\mathrm{MBBP}} = 11$ are applied. We set $I_{\max} = 100$ and $I_{\mathrm{sync,MBBP}} = 10$.

We observe in Figure 7.5 and Figure 7.6 that the number of stopping sets is a valuable performance criterion to predict the decoding performance on the AWGN channel. This is a consequence of the fact that family $\mathcal{F}_2$ has a poor performance while the performances of $\mathcal{F}_1$ and $\mathcal{F}_3$ are comparable. We report that the situation is similar if MBBP-X approaches are used. For the sake of a compact presentation, only the results of $\mathcal{F}_1$ and $\mathcal{F}_2$ are shown. The comparison of Figure 7.5 and Figure 7.6 allows to observe that all MBBP-NX approaches perform

significantly better than the MBBP-X schemes. More results in this direction can be found in [HHML07, HHML09].

7.5.2 Simulation Results for the $[127, 64, 21]$ BCH Code

We include simulation results for the $[127, 64, 21]$ BCH code for several reasons. First, we intend to show that MBBP compares favorably to BP if codes of larger dimension and co-dimension are used. Second, this code is a good example to demonstrate the capabilities of the MBBP-NX-FS approach. In order to emphasize this, we show the average number of iterations used by each of the decoders in the MBBP-NX setups and compare it to BP. We state that the results on the iterations are representative for all algebraic codes tested, including the ones shown in Appendix D. The MBBP-NX decoders obtain poor performance results and are omitted.

The $[127, 64, 21]$ BCH code has dimension 64 and co-dimension 63. It is computationally infeasible to find all minimum weight codewords of the dual code. Instead, we apply the algorithms provided in [HFE04] to identify a tight upper bound on the minimum distance of the dual code. This tool also allows us to generate a (most likely incomplete) set of codewords with weight equal to the estimated minimum distance of the dual code. We report 22 as an upper bound on the minimum distance of the dual code, and that at least 21 cog families exist. It is a very complex task to identify the number of stopping sets up to the size 21 in a code of length 127. We choose to use an approximation by evaluating only the number of stopping sets up to $\sigma = 5$. This allows us to conclude that all families yield comparable decoding performance. This observation is confirmed by simulation results.

Figure 7.7 shows the BER performance for different variants of the MBBP approach, where $l_{\mathrm{MBBP}} = 10$ decoders were run in parallel. There exist similar observations for the FER, but these results are not shown. Also given is the average number of iterations per decoder required for the convergence of different decoding approaches. We observe a significant performance improvement when comparing MBBP-NX and BP. However, for this code and other of comparable dimension, there remains a significant gap between the obtained performance and the Gallager bound. Increasing the variable l_{MBBP} further could close this gap but is considered as being impractical. The MBBP-NX-S approach uses more iterations per decoder on the average when compared to BP, what is due the fact that a decision from each decoder is required. Contrary, the MBBP-NX-FS requires only a decision from one decoder and consequently, the average number of iterations per decoder is even lower than that of standard BP.

As we are mainly interested in decoding performance at a given code length and only secondary in a good decoding performance at low complexity, we mainly focus on the MBBP-NX-S approach in the remainder of this work.

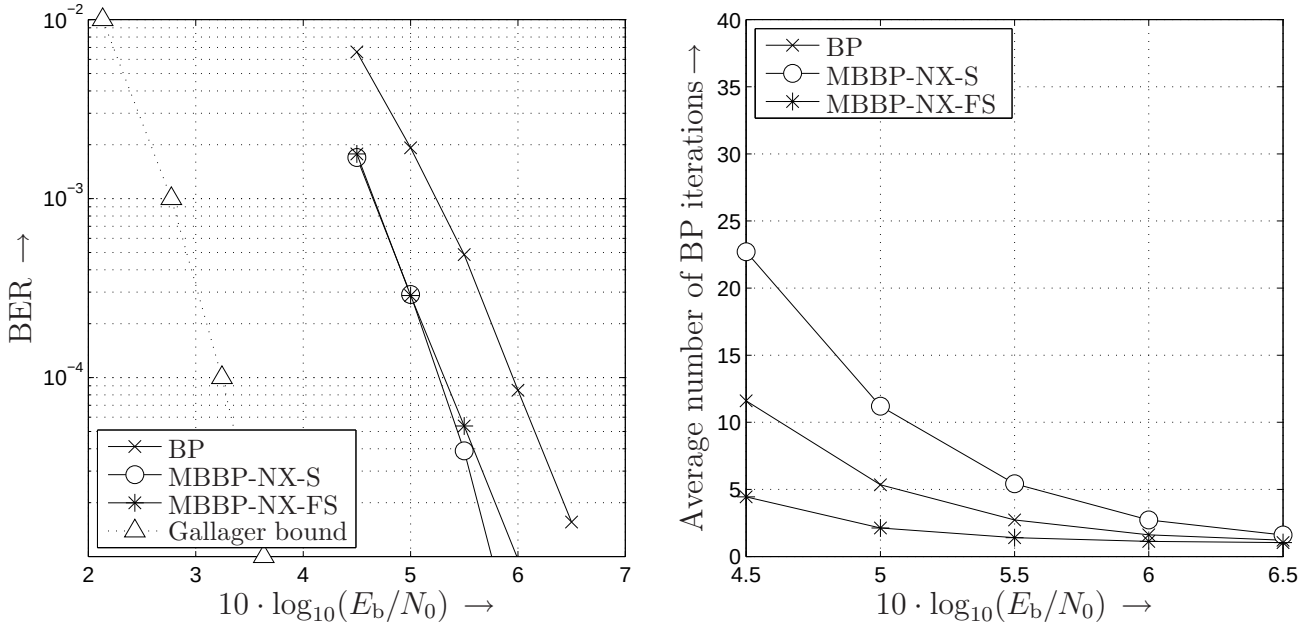


Figure 7.7: Performance comparison for the $[127, 64, 21]$ BCH code. The BP decoder and MBBP-NX decoders with $l_{\mathrm{MBBP}} = 10$ are applied. We set $I_{\max} = 100$.

7.6 The Leaking Algorithm

In this section, we introduce another approach to improve the decoding performance of an iterative decoding system. Contrary to the discussions before, we focus our attention now on modifications of a single BP unit. The decoding algorithm we are about to present implements a special scheduling principle which differs from the standard flooding scheduling scheme of BP decoding, cf. Section 4.2.2. Recently, several authors [CGW07, GA07] have proposed optimized scheduling schemes for BP decoding of LDPC codes. They report that these methods compare favorably to standard BP, in particular if a low number of iterations is used. A sophisticated scheme updating the nodes in ascending and descending order, respectively, was presented in [ZWFY07]. This allows for low error rates within parts of decoded word and is hence amenable for a parallel decoding setup. Some of the recently presented approaches choose a slightly different way. These approaches replace information at selected nodes after a certain number of iterations. Modifications of this type can be understood as a generalization of new scheduling features. In [VF04], sets of "unreliable" variable nodes are detected by their node degree and by the intrinsic information obtained via the channel. At these nodes, the input from the channel is replaced by saturation values. Multiple patterns of saturation values in the received word are tested for successful decoding of the codeword.

We introduce a promising new algorithm which modifies the data passed to the decoder. It estimates the reliability of the channel information for each encoded symbol and delays the entry of the unreliable channel information into the decoding process. The goal of this algorithm is to mitigate the effect of short cycles. We refer to it as the *leaking algorithm* (short: leaking) [He07, HHHL08]. The leaking algorithm is a valuable addendum for the MBBP approach. Although

intended for dense algebraic codes, it is remarkable that the application area of this scheme includes codes tailored for iterative decoding.

7.6.1 Basic Concepts and Motivation

Recall that short cycles are one of the crucial factors for preventing the decoder from successfully converging to the correct codeword, cf. Chapter 5. Especially algebraic codes have short loops in the Tanner graph that allow information to return to its source after a small number of iterations. Obviously, wrong information causes the decoder to converge to the wrong direction or stay undecided. The objective of the leaking algorithm is to reduce the influence of this feedback by instructing only subsets of the variable nodes with the information obtained from the channel. As the decoder cannot distinguish correct from erroneous channel information, it relies on the fact that for an AWGN channel information with a high reliability is more likely to be correct. The sets of instructed variables at iteration I, denoted by $\mathcal{L}_I \subseteq \{1, \ldots, n\}$, are chosen such that $\mathcal{L}_I \subseteq \mathcal{L}_{I+1}$ and such that the most reliable information is included first. Metaphorically speaking, information is kept from the decoder but slowly "leaks" into it, hence the naming. The benefit of the leaking algorithm is two-fold. First, it prevents unreliable information from being processed in the iterative algorithm within the first iterations. If the channel information does not fulfill the reliability criterion, the corresponding variable nodes are instructed that the channel reported $\Pr(X_i = 0 \mid Y_i = y_i) = 0.5$, $i \in \{1, \ldots, n\} \backslash \mathcal{L}_I$. This corresponds to erased variables when transmitting over the BEC. Recall from Section 6.2 that cycles are not a crucial factor for BEC decoding.

We show in the following that similar positive effects arise when using the leaking algorithm. Although the "erased" variable nodes are not informed about the channel output, they do take part in the decoding process, i.e. they update the probabilities associated with their variable node. With this, a multitude of erasures is included in the decoding process. The breaking of the cycles happens in early iterations and at the check nodes. Whenever a check node is connected to an erasure, this check node is not able to distribute information any longer, cf. Equation (3.26). Clearly, this slows down the convergence process, but may also lead to a better decoding result. A second advantage shows in later iterations. By then, many variable nodes have converged to a strong decision. Variable nodes that combine a set of incoming messages, where at least one of them is being close to one or zero, are almost unaffected by additional incoming information. As a consequence, unreliable information entering the decoder is strongly attenuated before returning to its origin by means of a loop. Of course, both correct as well as incorrect information is prevented from being forwarded and thus it would be advantageous to decode with the leaking algorithm and standard BP. Afterwards, the better solution could be picked. Consequently, the presented approach serves well as an additional representation in an MBBP setup. In that case, we denote the approach as *L-MBBP*.

We state that leaking is a suboptimal decoding approach as information is retained from the decoder for a given number of iterations. But as BP is suboptimal itself, this method is capable of improving the decoding performance, especially if the current graph has a high number of short loops. Figure 7.8 visualizes the leaking algorithm as a "semipermeable" wall that changes its aperture with increasing number of iterations.

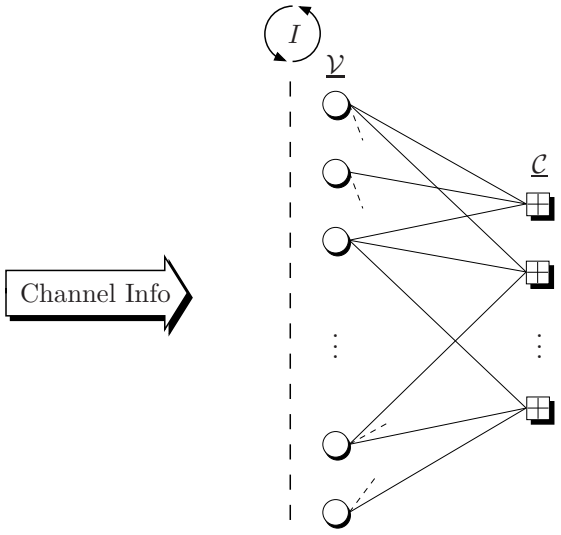


Figure 7.8: Visualization of the leaking algorithm as an iteration-dependent "semipermeable" wall.

Let us now explain the details of the proposed approach. A standard BP decoder passes all channel information to the variable nodes before the first iteration is carried out. Contrary, the leaking algorithm holds back less reliable parts of this information and lets it leak gradually into the variable nodes during the decoding process, where probabilities close to 0 and 1 are considered as being reliable. We assume that in iteration I, the set $\mathcal{L}_I$ of variable nodes is instructed about the corresponding channel output y_i, $i \in \mathcal{L}_I$. Assume that for iteration $I+1$ further intrinsic probabilities are scheduled to be included in the decoding process. That means that the set $\mathcal{L}_{I+1} \backslash \mathcal{L}_I \neq \emptyset$ of variable nodes need to be informed about the corresponding channel outputs. The inclusion is performed by a normalized multiplication of the current a-posteriori probability, which consists of a combination of extrinsic probabilities so far, and the probability now obtained from the channel. Once this operation is completed, each variable node may calculate the extrinsic probabilities which are sent out to the check nodes and resume the standard decoding process.

It still needs to be clarified how the information forwarded to the variable nodes is selected. Here, we follow two strategies, the *threshold-based approach* and the *probability-based approach*. We introduce the qualifiers (t) and (r) to distinguish between the approaches. For example, we use $\psi_I^{(\mathrm{t})}$ and $\psi_I^{(\mathrm{r})}$, respectively, to denote the set of informed variables for the different approaches at iteration I.

The threshold-based approach respects all probabilities exceeding a given reliability threshold. In the initialization phase, the decoder determines $\mathcal{L}_1^{(\mathrm{t})}$ by checking which channel probabilities exceed a given threshold $\psi_1^{(\mathrm{t})}$, $\psi_1^{(\mathrm{t})} \geq 0.5$, or underrun $1 - \psi_1^{(\mathrm{t})}$. How this threshold evolves with increasing number of iterations is defined below. We specify a second solution, the probability-based approach, for determining the variables included in the iterative decoding process by setting the probability $p_{\mathcal{L}}$ of variable nodes that are instructed on their intrinsic information. Obviously, this ratio corresponds to a threshold $\psi_1^{(\mathrm{r})}$, $\psi_1^{(\mathrm{r})} \geq 0.5$, such that the channel information is leaked to the set of variable nodes with indices $\mathcal{L}_i^{(\mathrm{r})}$, where

$$\mathcal{L}_1^{(\mathrm{r})} = \{i \mid \max\left(\Pr(X_i = 0 \mid Y_i = y_i), \Pr(X_i = 1 \mid Y_i = y_i)\right) > \psi_1^{(\mathrm{r})}\},\ i = 1, \ldots, n.$$

It is straightforward to determine $p_{\mathcal{L}}$ as a function of $\psi_1^{(\mathrm{r})}$. Observe first that for a given probability-domain threshold $\psi_1^{(\mathrm{r})}$, all received values y_i exceeding a threshold $\psi^{(\mathrm{y})}$ with

$$|y_i| \geq \psi^{(\mathrm{y})} = \frac{\sigma_{\mathrm{R}}^2}{2} \ln\left(\frac{\psi_1^{(\mathrm{r})}}{1 - \psi_1^{(\mathrm{r})}}\right), \tag{7.7}$$

are passed as channel output information to the decoding process. Elsewise, an erasure is declared. Recall that σ_{R}^2 is the power of the real-valued noise, which is relevant for BPSK transmission, cf. Chapter 2. Thus,

$$p_{\mathcal{L}} = \Pr(Y \leq -\psi^{(\mathrm{y})}) + \Pr(Y \geq \psi^{(\mathrm{y})}) = \mathrm{Q}\left(\frac{\psi^{(\mathrm{y})} - 1}{\sigma_{\mathrm{R}}}\right) + \mathrm{Q}\left(\frac{\psi^{(\mathrm{y})} + 1}{\sigma_{\mathrm{R}}}\right) \tag{7.8}$$

holds. Using Equation (7.7) and Equation (7.8), one obtains

$$p_{\mathcal{L}} = \mathrm{Q}\left(\frac{\sigma_{\mathrm{R}}}{2} \ln\left(\frac{\psi_1^{(\mathrm{r})}}{1 - \psi_1^{(\mathrm{r})}}\right) - \frac{1}{\sigma_{\mathrm{R}}}\right) + \mathrm{Q}\left(\frac{\sigma_{\mathrm{R}}}{2} \ln\left(\frac{\psi_1^{(\mathrm{r})}}{1 - \psi_1^{(\mathrm{r})}}\right) + \frac{1}{\sigma_{\mathrm{R}}}\right). \tag{7.9}$$

For a given SNR, Equation (7.9) can be solved for $\psi_1^{(\mathrm{r})}$ prior to decoding.

Both $\psi_1^{(\mathrm{r})}$ and $\psi_1^{(\mathrm{t})}$ are known at this point. With increasing iteration number, the threshold is decreased linearly by

$$\psi_I^{(\mathrm{t/r})} = \max\left(\psi_1^{(\mathrm{t/r})}\left(1 - \frac{I-1}{I'_{\max}}\right), 0.5\right),\ 1 < I \leq I_{\max},$$

where $I'_{\max}$ is a user parameter that sets the slope of the threshold function. The notation $\psi_I^{(\mathrm{t/r})}$ states that this coherence holds for both $\psi_I^{(\mathrm{t})}$ and $\psi_I^{(\mathrm{r})}$. Simulations have shown that it is advantageous if $I'_{\max}$ is chosen such that (almost) all channel information is included at the end of the decoding process. A description of the leaking algorithm in form of pseudocode is given in Algorithm 7.3.

Algorithm 7.3 Leaking Algorithm.

Input: $\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, $I'_{\max}$, as well as $\psi_1^{(\mathrm{t})}$ or $p_{\mathcal{L}}$
Output: $\hat{\boldsymbol{x}}$
Function call: $\hat{\boldsymbol{x}}$=leaking($\boldsymbol{y}$, $\boldsymbol{H}$, $I_{\max}$, $I'_{\max}$, as well as $\psi_1^{(\mathrm{t})}$ or $p_{\mathcal{L}}$)

/* Initialization Phase */

Calculate vector $\psi_I^{(\mathrm{t/r})}$ from $I'_{\max}$ and $\psi_1^{(\mathrm{t})}$ or $p_{\mathcal{L}}$, $I = 1, \ldots, I_{\max}$

/* Processing Phase */

for $I = 1, \ldots, I_{\max}$ **do**
  for $i = 1, \ldots, n$ **do**
    if $p(X_i = 0 \mid Y_i = y_i) \geq \psi_I^{(\mathrm{t/r})}$ **or** $p(X_i = 0 \mid Y_i = y_i) \leq 1 - \psi_I^{(\mathrm{t/r})}$ **then**
      $p_i^{\mathrm{int}} = \Pr(X_i = 0 \mid Y_i = y_i)$ **/* Include channel info in decoding process */**
    else
      $p_i^{\mathrm{int}} = 0.5$ **/* Exclude channel info */**
    end if
  end for
  $[p_i^{\mathrm{app}}, \mathrm{success\_flag}, p_{ij}^{\mathrm{ext}}] = \mathrm{bp\_decode}(p_i^{\mathrm{int}}, \boldsymbol{H}, 1, p_{ij}^{\mathrm{ext}})$ $i = 1, \ldots, n$, $j \in \mathcal{N}(\mathrm{v}_i)$
  if success_flag == 1 **then**
    $\hat{\boldsymbol{x}} = \lfloor \boldsymbol{p}^{\mathrm{app}} \rceil$, **break**
  end if
  $I = I + 1$
end for

7.6.2 Simulation Results for L-MBBP Decoding

We present simulation results which indicate that the application of L-MBBP leads to performance improvements. In this example, we use the probability-based approach. The $[63, 39, 9]$ BCH code is a prime example for this algorithm. The code dual to the considered one has minimum weight 14 and contains only 450 codewords of this weight, resulting in 6 COGs, all from one family. Creating an MBBP setup with $l_{\mathrm{MBBP}} = 6$ representations is straightforward. However, the task of increasing l_{MBBP} further is difficult as it requires either more codewords of the dual code, or alternative decoders, such as the realization by the leaking algorithm. The lack of additional representations is an important issue, especially for constructed LDPC codes which do not have a rich structure. For the $[63, 39, 9]$ BCH code we show that using codewords from the dual with next-higher weight (which is 18) is not a favorable approach. Decoders running on matrices constructed from these codewords obtain a significantly poorer performance. Contrary, the leaking algorithm in connection with minimum-weight codewords of the dual code provides performance results which are even slightly better than standard BP. As to be observed later, this fact also holds for constructed codes, e.g. PEG codes. Another possibility of the L-MBBP setup is presented exemplary for this code. Running decoders with different values for $p_{\mathcal{L}}$ allows for a further increase of representations and better decoding performance. Figure 7.9 shows simulation results for both the BER and the FER. For this simulation, $I_{\max} = 100$, $I'_{\max} = 300$, and the probabilities for instructed variables $p_{\mathcal{L},1} = 0.98$, $p_{\mathcal{L},2} = 0.95$, $p_{\mathcal{L},3} = 0.9$, $p_{\mathcal{L},4} = 0.85$ are set.

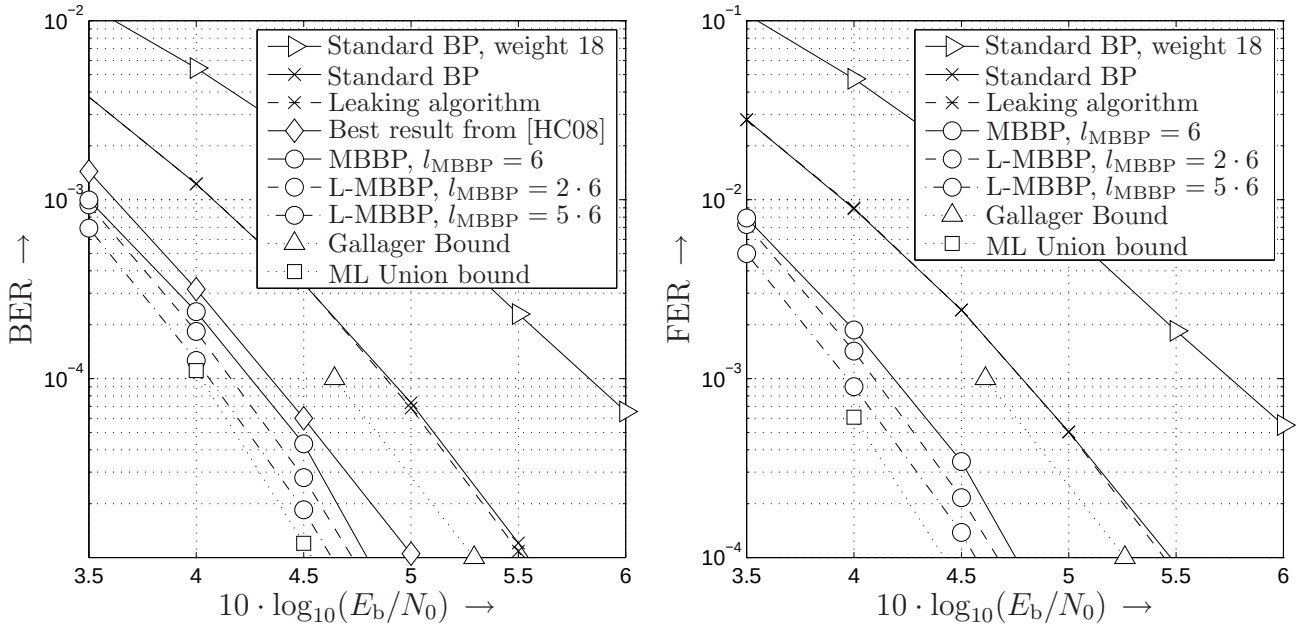


Figure 7.9: BER and FER performance, $[63, 39, 9]$ BCH. We set $I_{\max} = 100$ and $p_{\mathcal{L},1} = 0.98$, $p_{\mathcal{L},2} = 0.95$, $p_{\mathcal{L},3} = 0.9$, and $p_{\mathcal{L},4} = 0.85$.

It can be observed that significant performance improvements of about 0.75 dB can be obtained by using L-MBBP instead of BP. Henceforth, we use this type of algorithm in our investigations.

7.7 L-MBBP Decoding of LDPC Codes

In this section, we focus on the applicability of the multiple-bases approach to LDPC codes, including constructed codes and codes from the WiMAX standard. Let us first consider MBBP decoding of LDPC codes constructed by the PEG algorithm. Contrary to algebraic codes, these codes have no rich mathematical structure. The PEG algorithm follows a recipe to construct LDPC codes. In this process, it uses the decisions of (pseudo)-random number generators and consequently, the existence of a large automorphism group cannot be guaranteed. Attempts to identify this group by computer search with the *Magma* software package resulted in a set containing only the identity permutation. For this reason, it is more difficult to provide a set of unequal matrices with comparable decoding performance. One option is to identify minimum-weight codewords of the dual code that are not used in the matrix constructed by the algorithm. These codewords can be used to replace existing parity-check equations in the given matrix. However, the PEG algorithm tends to use all the words of low weight in the constructed matrix. To substantiate this, all codewords of the dual of PEG codes with rate 1/2 and length up to 50 were found. From this set, all codewords of lowest and second-lowest weight were identified as the codewords used in the constructed parity-check matrix. Furthermore, finding these codewords becomes computationally infeasible with increasing code dimension.

Another option to create additional matrix representations would be to generate linear combinations of existing parity checks randomly. This approach usually results in matrices with very poor decoding performance, what can be attributed to the high density of the matrix. We propose an algorithm which makes use of the fact that the shortest cycles of the considered PEG codes are of length 4 and 6, cf. Table 5.2. This allows to identify linear combinations of parity-check equations with weight comparable to the ones created by the PEG algorithm. Once a given number of redundant checks is found, they are used to replace original checks what leads to a new matrix. Note that this process does not change the properties of the code, in particular not the rate. A detailed explanation is provided below.

7.7.1 Additional Parity Checks for PEG Codes

The proposed method of finding additional parity-check equations for PEG codes is discussed in detail. We start by introducing the required notation. We denote the j-th set of indices of check nodes (i.e. rows of $\boldsymbol{H}$) which are connected to each other by at least one cycle of length c from $\mathcal{T}_{c,j}$, where $\mathcal{T}_{c,j}$ is the j-th set of check-node indices connected by a cycle of length c, $j \in \{1, \ldots, |\mathcal{T}_c|\}$, and $|\mathcal{T}_c|$ denotes the number of these sets. We create additional parity checks by finding linear combinations of given parity-check equations which are connected by a cycle. The linear combination cancels 1-entries that are common in an even number of parity-check equations. The fact that the parity checks are connected by a girth assures that the checks share at least two 1-entries. In the following, we establish an upper bound on the weight of the resulting check. The weight of a linear combination of $c/2$ checks indexed by the elements in $\mathcal{T}_{c,j}$ can be upper bound by

$$\omega_{\mathrm{r}} \leq \sum_{i \in \mathcal{T}_{c,j}} \mathrm{wt}(\boldsymbol{H}(i,:)) - c. \tag{7.10}$$

Such a check can replace a check indexed by one element of $\mathcal{T}_{c,j}$ in $\boldsymbol{H}$ while the matrix keeps full rank. Figure 7.10 sketches how a redundant check for $c = 4$ is generated from $c/2 = 2$ existing equations. We report from computer simulations that the upper bound in Equation (7.10) is usually taken. In the following, we apply these steps multiple times to replace a subset of the parity-check equations present in $\boldsymbol{H}$. This number depends on the local girth distribution of the originally constructed parity-check matrix. Usually, 10 to 20 percent of the rows are replaced to generate a new matrix representation. With this method we generate MBBP setups for decoding PEG codes of length up to $n = 1000$. For $n \leq 200$, we use local cycles of length $c = 4$ and $c = 6$, while cycles of length $c = 6$ are used for the remaining lengths.

At this point, we only provide a sample of the results obtained with this method. More results on L-MBBP decoding of PEG codes are presented in Chapter 8. In the present section, we provide results for a PEG code of length $n = 200$ and rate 1/2. For the leaking algorithm and L-MBBP the probability-based approach with $p_{\mathcal{L}} = 0.9$ was used.

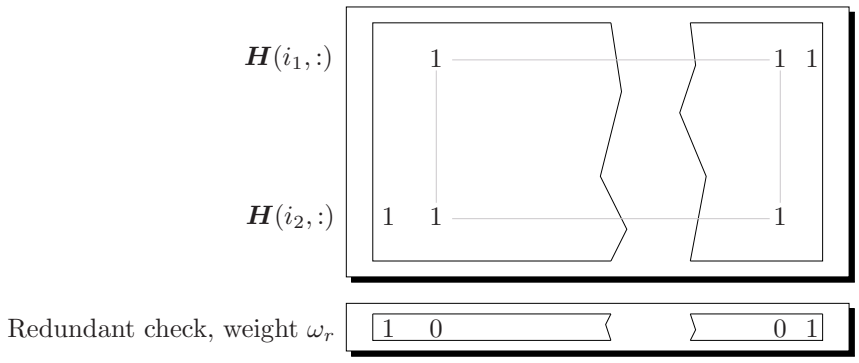


Figure 7.10: Construction of a redundant check equation for $c = 4$ with $i_1,\ i_2 \in \mathcal{T}_{cj},\ j \in \{1, \ldots, |\mathcal{T}_c|\}$.

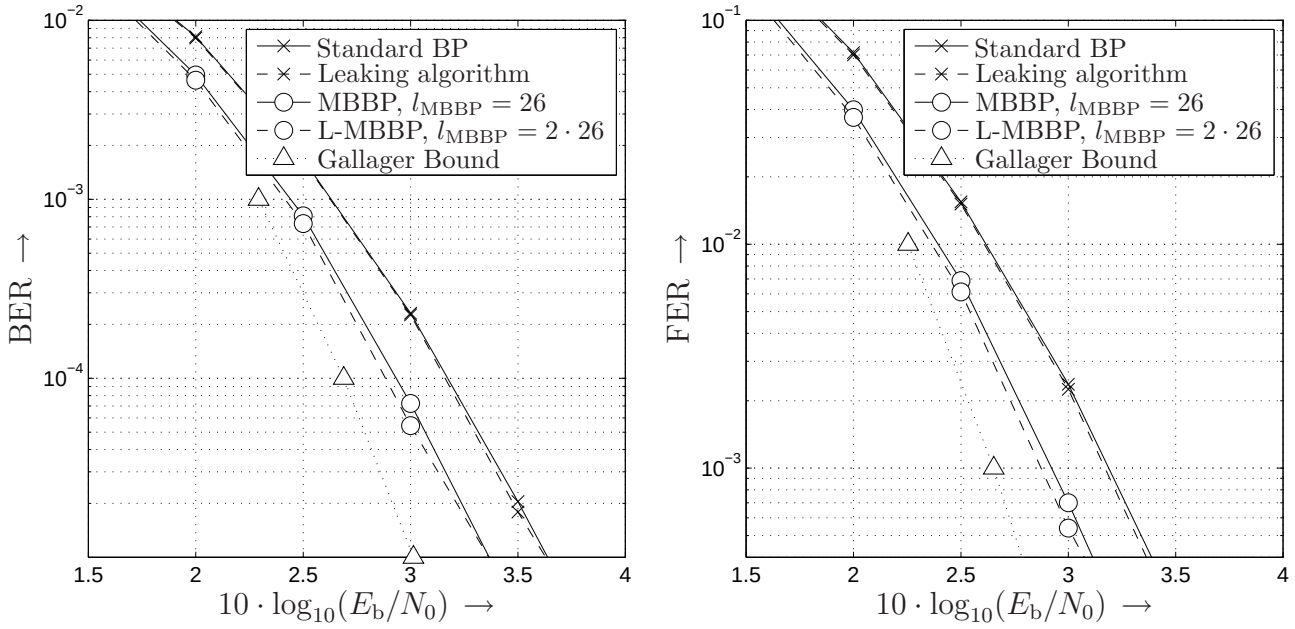


Figure 7.11: BER and FER performance, $[200, 100]$ PEG. We set $I_{\max} = 200$ and $p_{\mathcal{L}} = 0.9$.

Using the L-MBBP approach, performance improvements of around 0.3 dB are obtained, cf. Figure 7.11. This is a very remarkable result for two reasons. First, this method closes about half the gap to the Gallager bound at $n = 200$. Second, PEG codes achieve in general outstanding performance results when a short or moderate codelength is required. To the best knowledge of the author, no other approaches obtaining a comparable performance improvement with a given PEG code are currently known. We report from simulations that increasing the number of iterations to $I_{\max} > 200$ is not an alternative to this approach as it does not lead to a performance improvement.

7.7.2 Parity-Check Matrices for WiMAX LDPC Codes

Channel codes defined in telecommunications standards, e.g. by the IEEE or the ETSI, usually show a rich structure in the graph, what allows for storing the matrix on devices with small memory. We make use of these structural properties in order to generate redundant parity-check equations which show good decoding properties. This method compares favorably to applying the approach from Section 7.7.1 to this class of codes, which would be a straightforward solution for this task. As stated in Section 5.1.2, we focus on rate-1/2 channel codes from the WiMAX standard with $576 \leq n \leq 960$. These lengths correspond to the expansion factor w, $24 \leq w \leq 40$, cf. Section 5.1.2.

The LDPC codes used in the WiMAX standard belong to the group of *lifted codes* [Tan81]. The parity-check matrices of these codes are created as follows. A base matrix, in general non-binary, is provided and its entries are partitioned into two sets. In order to create the parity-check matrix, one replaces all entries from the first set by a permutation matrix of size $w \times w$. The deployed permutation matrix depends on the corresponding entry in the base matrix. Further, all entries from the second set are to be replaced by a zero-matrix, also of size $w \times w$. Note that we assume that the resulting matrix is binary.

Let us now specialize to the lifted codes from the WiMAX standard. In the following, the variable $\boldsymbol{H}'_{\mathrm{b}}$ specifies the non-binary base matrix and $\boldsymbol{H}$ the binary parity-check matrix for all rate-1/2 LDPC codes in [IEE05, p. 628]. The matrix $\boldsymbol{H}'_{\mathrm{b}}$ is fixed and reads

$$\boldsymbol{H}'_{\mathrm{b}} = \begin{pmatrix}
-1 & 94 & 73 & -1 & -1 & -1 & -1 & -1 & 55 & 83 & -1 & -1 & 7 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 27 & -1 & -1 & -1 & 22 & 79 & 9 & -1 & -1 & -1 & 12 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 24 & 22 & 81 & -1 & 33 & -1 & -1 & -1 & 0 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
61 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & 65 & 25 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 39 & -1 & -1 & -1 & 84 & -1 & -1 & 41 & 72 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 46 & 40 & -1 & 82 & -1 & -1 & -1 & 79 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 95 & 53 & -1 & -1 & -1 & -1 & -1 & 14 & 18 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & 11 & 73 & -1 & -1 & -1 & 2 & -1 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
12 & -1 & -1 & -1 & 83 & 24 & -1 & 43 & -1 & -1 & -1 & 51 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 94 & -1 & 59 & -1 & -1 & 70 & 72 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & 7 & 65 & -1 & -1 & -1 & -1 & 39 & 49 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
43 & -1 & -1 & -1 & -1 & 66 & -1 & 41 & -1 & -1 & -1 & 26 & 7 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{pmatrix}.$$

The base matrix $\boldsymbol{H}'_{\mathrm{b}}$ is only used directly for constructing the binary parity-check matrix if the expansion factor $w = 96$ is used. In general, a prior normalization of all positive entries is done before the lifting process described above is invoked. The matrix resulting from the normalization process is denoted as $\boldsymbol{H}_{\mathrm{b}}$, where

$$H_{\mathrm{b}}(i,j) = \begin{cases} \left\lfloor \frac{H'_{\mathrm{b}}(i,j)\cdot w}{96} \right\rfloor & \text{if } H'_{\mathrm{b}}(i,j) > 0, \\ H'_{\mathrm{b}}(i,j) & \text{if } H'_{\mathrm{b}}(i,j) \leq 0 \end{cases} . \tag{7.11}$$

We describe how the binary parity-check matrix $\boldsymbol{H}$ is constructed from $\boldsymbol{H}_{\mathrm{b}}$. Here, the partitioning into two sets distinguishes negative entries from non-negative entries. Negative entries (constant value -1) need to be replaced by a $w \times w$ zero-matrix. All entries $H_{\mathrm{b}}(i,j) \geq 0, \forall\, i,j$ are to be replaced by an identity matrix which is cyclically shifted to the right by $H_{\mathrm{b}}(i,j)$ positions. Note that this includes entries of value 0 which are to be replaced by a standard identity matrix.

7.7.3 Additional Parity Checks for WiMAX LDPC Codes

We introduce an approach to find well-performing redundant parity checks. The core idea of this algorithm is to utilize the structure of the base matrix of the code and generate redundant checks for $\boldsymbol{H}_{\mathrm{b}}$ instead of $\boldsymbol{H}$. Subsequently, a selection of these checks is lifted. With this we obtain redundant checks of the binary parity-check matrix $\boldsymbol{H}$. We create redundant parity-check equations from $\boldsymbol{H}_{\mathrm{b}}$ by adding two checks which do not share positive elements in a column. This is for example the case in rows 3 and 4 of $\boldsymbol{H}_{\mathrm{b}}$ (see $\boldsymbol{H}'_{\mathrm{b}}$ to verify this). Although the weights of the parity checks add, redundant equations of low weight are possible. A linear combination of rows 11 and 12 results in a redundant check of weight 10. Note that such a check can be used for all the considered WiMAX codes and results in w possibly redundant checks for $\boldsymbol{H}$. Depending on the length of the code, we replace 10 to 16 rows in the existing binary parity-check matrix in order to generate a new representation. In this context, every redundant check is individually positioned in such a way that full rank is preserved.

Let us now compare the proposed method to the general approach from Section 7.7.1. Considering that any entry in $\boldsymbol{H}_{\mathrm{b}}$ is to be replaced by a permutation matrix (row weight one), it is easy to see from $\boldsymbol{H}_{\mathrm{b}}$ that each row-weight of $\boldsymbol{H}$ is 6 or 7, regardless of the actual length n of the code. Exemplary, we use the shortest and most dense rate-1/2 code of length $n = 576$. For this code, the local girth distribution of $\boldsymbol{H}$ contains $c = 6$ and $c = 8$. Using Equation (7.10) and $c = 6$, it can be deducted that additional parity-check representations have a weight of at most 12 to 15. As stated in Section 7.7.1, this bound is met by the realizations. Contrary, the proposed method obtains a row weight of 10. For codes of higher length, the method from Section 7.7.1 is challenged even more, whereas the results deducted from the base matrix do not vary with n. It is worth mentioning that the algorithms from [HFE04] did not find redundant codewords in the dual of this code with a weight lower than 10.

Again, we provide only a sample of the available results and refer the reader to Chapter 8 for more results of L-MBBP decoding of codes from the WiMAX standard. We present results for the code of length $n = 576$ ($w = 24$). In our simulation, the decoders in the L-MBBP setup are allowed to perform 200 iterations at most. We use $l_{\mathrm{MBBP}} = 15$ MBBP realizations and also

apply the leaking algorithm, where we set $p_{\mathcal{L}} = 0.9$. Figure 7.12 shows simulation results for both the BER and FER.

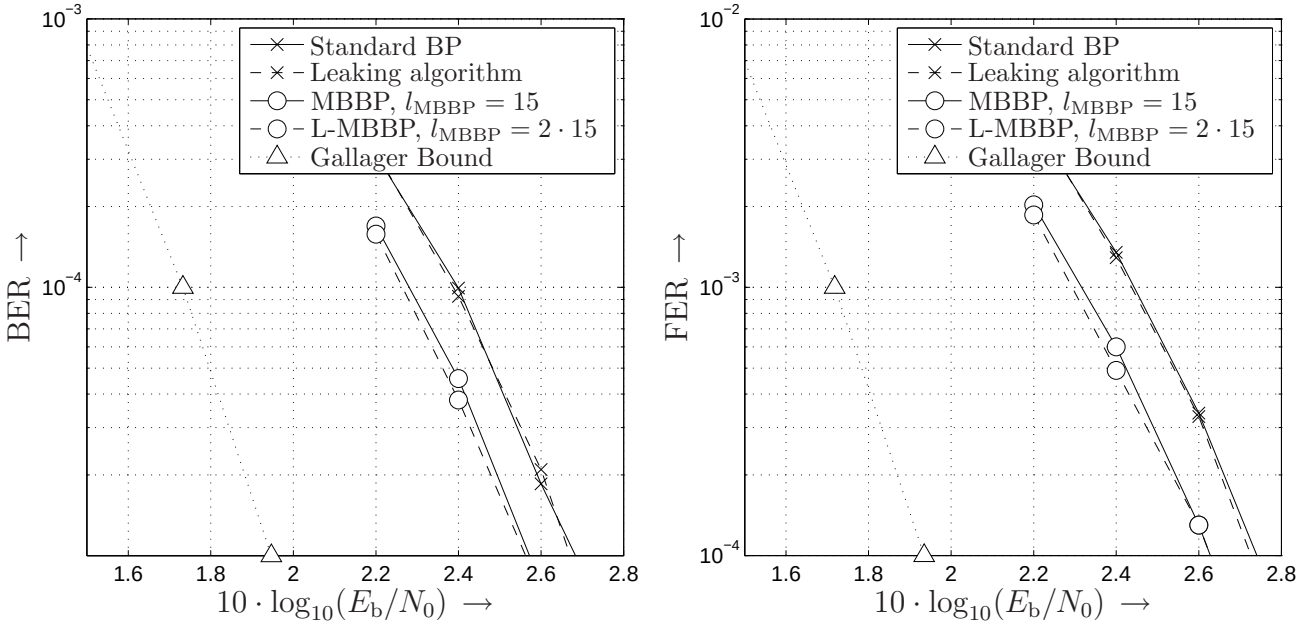


Figure 7.12: BER and FER performance, $[576, 288]$ code from the IEEE 802.16 standard. We set $I_{\max} = 200$ and $p_{\mathcal{L}} = 0.9$.

It is to observe that a gain of around 0.15 dB at BER $= 10^{-5}$ can be achieved by using the L-MBBP approach. Chapter 8 shows that improvements of this type can be obtained for all WiMAX LDPC codes considered in this work. As a remark we state that a slight modification of the presented approach leads to comparable results but shows further advantages. Instead of replacing single rows in $\boldsymbol{H}$, groups of w rows, which correspond to one row of $\boldsymbol{H}_{\mathrm{b}}$, are replaced. The described variation keeps the structural properties of the original matrix and hence allows for efficient storing of the additional matrix representations.

Chapter 8

Decoding Performance and Delay of Current Channel Coding Schemes

One of the costs for the gain in power efficiency by means of channel coding is the increased delay of transmitted data. Generally, the delay due to transmission is the time difference between the information being emitted by the source and being delivered to the sink. This delay consists of three parts. The *structural delay*, to be defined in detail in this chapter, is the delay that is inevitable when a channel coding system is used and when it can be assumed that the delay due to limited processing is negligible. Contrary, the *processing delay* depends only on the computational power of the involved data processors and the quality of the implementation. The third part is specified by the time required for wave propagation. We discuss both structural delay and processing delay in this thesis[1]. We show that the structural delay is already very important in current transmission systems and will become even more important in the future. These results allow us to contrast convolutional codes with block codes in order to determine favorable schemes when good decoding performance and a low delay are required at the same time. It is shown in this chapter that convolutional codes obtain a very good trade-off in this scenario. For the competing block codes, the BP and the L-MBBP approach are taken into consideration. Especially the latter approach is tailored for this application, as its parallel structure allows to improve the performance while keeping the delay constant. All investigations in this context are done under the assumption that the data is signaled over the AWGN channel.

8.1 Structural Delay

First, we provide a general definition for the structural delay [HH09a] and subsequently specialize it for convolutional codes and block codes. At this moment, additional delay caused by limited performance of processing units is not respected. In other words, data processors with infinite speed are assumed. This assumption does not affect the transmission parameters, i.e. the average

[1]The delay due to wave propagation depends mainly on the distance between transmitter and receiver. This is not a communications engineering problem and is hence ignored.

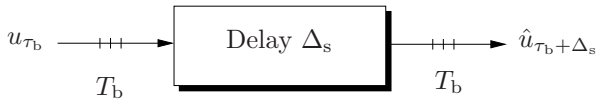


Figure 8.1: Schematic representation of a transmission setup by a delay unit (structural delay only).

transmission time per bit, T_b and the average transmission time per encoded symbol, T_s, cf. Chapter 2. As the source is assumed to be binary and free of redundancy, cf. Chapter 2, page 9, one bit of information is transported by one source symbol. The signaling speeds are connected to each other by the overall rate of the transmission system as we assume a cache-free data flow through all processing devices. Let us introduce additional variables that allow us to denote the sampling instants properly. By u_{τ_b} we denote the source symbol transmitted at time step τ_b. Figure 2.6 shows a block diagram of the transmission and provides a snapshot of all variables at time step i. In order to define the structural delay, we restrict our attention to the source symbols and the estimated source symbols which appear after the decoding process. Figure 8.1 shows a schematic representation of the end-to-end transmission scenario, where we consider a source variable and its delayed estimation at the receiver output. We provide a precise definition of the structural delay in the following.

Definition 8.1.1 *Assume that there exists a transmission setup which causes no delay due to wave propagation or limited computational performance of data processing units. In other words, all used processors are assumed to have infinite computational power and all transmission lines are infinitely short. Still, an infinitely fast transmission is not possible due to limited clock rates and the application of channel coding schemes. We define the number of time steps of duration T_b between the appearance of a source symbol and its delivery to the sink as the structural delay Δ_s of the considered channel coding scheme.*

Obviously, the structural delay is a lower bound on the actual delay which depends on the computational power in the system. The latter is referred to as the processing delay Δ_p and will be introduced in Section 8.3. We emphasize that the structural delay is a very important measure in the field of channel coding. From simulation experience on standard desktop computers we state that up to data rates of 8 MBit/s the structural delay and processing delay lie in a comparable range. Dedicated hardware platforms allow for even higher data rates. The importance of this type of delay will further increase with the progress in microelectronics. We also motivate this study by pointing out that this type of delay is a feature of the coding scheme itself, regardless of a current or future way of implementation.

Let us first discuss the structural delay of block codes, denoted by $\Delta_{s,bc}$ and then proceed to the structural delay of convolutional codes, denoted by $\Delta_{s,cc}$. For block codes, one crucial factor is the filling and depleting of the encoder and decoder, respectively. Thus, the structural delay depends significantly on how fast and in which order the data can be accessed. For that reason,

we distinguish between three transmission scenarios that occur very often in practice and span a wide range of delays.

8.1.1 Structural Delay of Block Codes

In this section, we discuss the structural delay of $[n, k, d]$ block codes of rate R with $n > 1$. We first focus on the case of serial processing (SP), i.e. it is assumed that the source delivers data in a symbol-by-symbol manner and with a rate of $1/T_\mathrm{b}$. This is a typical scenario for transmission of live streams. Serial processing also implies that the sink accepts data at a rate of $1/T_\mathrm{b}$. In this case, we have to distinguish between non-systematic encoding and systematic encoding. It turns out that the latter approach is advantageous in terms of decoding performance and structural delay [HH09a].

Subsequently, block processing (or parallel processing) of the data is discussed. This scenario resembles the practical situation of transmitting existing data on request, e.g. taken from a very fast memory device such as a cache memory. At the receiver side, a fast memory resembling the sink is assumed as well. This method imposes the lowest structural delay when block codes are used. As indicated in Section 5.1.3, systematic encoding compares favorably to non-systematic encoding with respect to decoding performance. Consequently, we restrict our attention to systematic encoding when block processing is used. This results in a scheme with good decoding performance and low structural delay. Recall that we look at BPSK signaling which causes no data latency due to mapping operations. The generalization to other modulation formats is straightforward.

Serial Processing with Non-Systematic Encoding

If serial processing with non-systematic encoding (SPNSE) is used, the codeword cannot be calculated before the encoder is informed about all $k = R \cdot n$ source symbols. This is a first source of structural delay. On the receiver side, structural delay is created by depleting the receiver in a serial manner. Figure 8.2 shows how the encoder is filled and the decoder is depleted. In this coherence, we assume that the channel decoder consists of a codeword estimator and a unit performing the inversion of the encoding process, cf. Figure 2.6. Hence it emits estimates for the sourceword of length k. Besides, a third source of structural delay is the transmission of

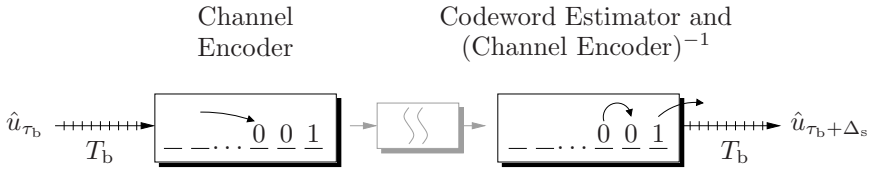


Figure 8.2: The encoder and decoder are filled and depleted, respectively, both in a serial manner and at a rate of $1/T_\mathrm{b}$.

the codeword itself at a rate of $1/T_\mathrm{s}$. This kind of delay does not resemble the delay due to

wave propagation. Instead, we respect the fact that the encoded symbols are mapped on basic impulses (PAM) and leave the transmitter at a rate $1/T_\mathrm{s}$. This rate is limited as regulations apply.

In order to calculate the delay, we analyze the transmission of one single codeword. A source symbol presented to the encoder at sampling instant τ_b^*, $\tau_\mathrm{b}^* \in \{1, \ldots, k\}$, is not processed before $k-1-(\tau_\mathrm{b}^*-1)$ further source symbols are readily available. Its corresponding estimate encounters a delay of $\tau_\mathrm{b}^* - 1$ time steps of duration T_b at the decoder as $\tau_\mathrm{b}^* - 1$ source symbols are depleted before the current source symbol interest. This sums to a delay of $k - 1 = R \cdot n - 1$ time steps of duration T_b. Furthermore, the transmission of the complete codeword at a clock rate of $1/T_\mathrm{s}$ takes $n - 1$ time steps of duration T_s longer than the transmission of an uncoded symbol. With $(n-1) \cdot T_\mathrm{s} = (n-1) \cdot R \cdot T_\mathrm{b}$, a total structural delay of

$$\Delta_{\mathrm{s,bc,SPNSE}} = 2 \cdot R \cdot n - R - 1, \quad n > 1,$$

time steps of duration T_b results for SPNSE of block codes of length n.

Serial Processing with Systematic Encoding

We discuss the case of serial processing with systematic encoding (SPSE). Systematic encoding is usually applied in the field of channel coding. Our investigations show two reasons for this. First, this kind of encoding leads to better decoding performance. Second, the structural delay in the encoder can be reduced, what is accomplished by choosing a generator matrix which contains the identity matrix within the first k columns. This allows to initiate the transmission of the encoded word prior to having received all source symbols. To be precise, we initiate the transmission of the source symbols as soon as $n \cdot R \cdot f_\mathrm{R}$ source symbols are present. Here, f_R is a factor depending on the rate of the channel code. We are interested in a continuous transmission. To this end, let us consider the transmission of the f_R-th part of a block of k source symbols. On the channel side, this transmission requires $n \cdot R \cdot f_\mathrm{R}$ time steps of length T_s. To avoid discontinuation, the f_R-th share of the remaining source symbols, i.e. $n \cdot R \cdot (1 - f_\mathrm{R}) \cdot f_\mathrm{R}$ further source symbols, have to be accepted at the channel encoder input during transmission of $f_\mathrm{R} \cdot n \cdot R$ source symbols. Respecting the signaling frequencies T_b and T_s,

$$n \cdot R \cdot (1 - f_\mathrm{R}) \cdot f_\mathrm{R} \cdot T_\mathrm{b} = n \cdot R \cdot f_\mathrm{R} \cdot T_\mathrm{s}$$

has to hold. Thus,

$$n \cdot R \cdot (1 - f_\mathrm{R}) \cdot f_\mathrm{R} \cdot T_\mathrm{b} = n \cdot R \cdot f_\mathrm{R} \cdot R \cdot T_\mathrm{b}$$

needs to be true. The non-trivial solution of this equation reads $f_\mathrm{R} = 1 - R$. Again, we discuss the transmission of one codeword to determine the structural delay. Let τ_b^*, $\tau_\mathrm{b}^* \in \mathbb{Z}$, denote the

sampling instant in which the considered source symbol has been generated. We assume again that we use a block code of rate R. The whole codeword is available at the receiver after the time interval $n \cdot R \cdot (1-R) \cdot T_\mathrm{b} + n \cdot T_\mathrm{s} = n \cdot R \cdot (1-R) \cdot T_\mathrm{b} + n \cdot R \cdot T_\mathrm{b} = n \cdot R \cdot (2-R) \cdot T_\mathrm{b}$ has passed. Figure 8.3 shows these coherences. It is to observe from Figure 8.3 that a source symbol

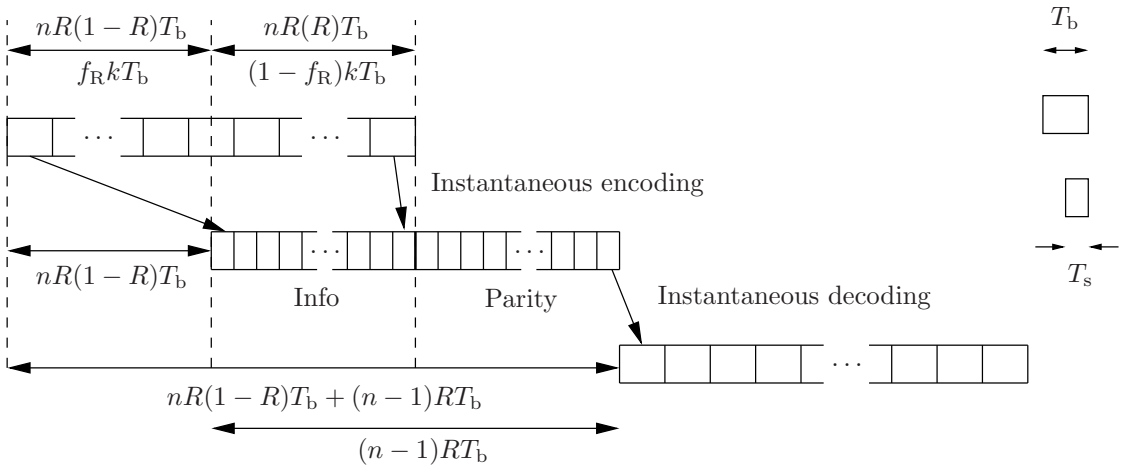


Figure 8.3: Delays for systematic encoding, shown for a code of rate 1/2.

transmitted at time step τ_b^* has a delay of $n \cdot R \cdot (1-R) \cdot T_\mathrm{b} + (n-1) \cdot R \cdot T_\mathrm{b} - \tau_\mathrm{b}^* = n \cdot R \cdot (2-R) - R - \tau_\mathrm{b}^*$ time steps of duration T_b before the decoding process can start. Additionally, assume that the delay for depleting the decoder imposes another delay of $(\tau_\mathrm{b}^* - 1)$ time steps of duration T_b. The overall structural delay for SPSE is given by

$$\Delta_{\mathrm{s,bc,SPSE}} = ((2-R) \cdot R \cdot n - R - 1), \quad n > 1,$$

time steps of duration T_b.

Remark 8.1.2 *Another way to obtain this result is to make further use of the delay of filling and depleting from SPNSE but counting the delay due to transmission only after the whole sourceword has been obtained, i.e. shortening the delay due to channel transmission for $n \cdot R \cdot R$ time steps T_b.*

Block (Parallel) Processing

The lowest possible delay is obtained for the special case of block processing (BlP), as no delay due to filling and depleting has to be respected. Instead, we assume that a very fast memory device passes the whole source word to the channel encoder within one sampling instant and the encoder is capable of processing these inputs in a parallel manner. In this case, the choice between non-systematic and systematic encoding has no influence on the structural delay. Resulting, we obtain a delay of $n-1$ time steps of duration T_s for the transmission of the block. The variable T_b is not relevant for block processing. Instead, we specify this structural delay as multiples of $\frac{1}{R} \cdot T_\mathrm{s}$ and obtain

$$\Delta_{\mathrm{s,bc,BlP}} = R \cdot (n-1), \quad n > 1,$$

time steps of duration $\frac{1}{R} \cdot T_{\mathrm{s}}$.

8.1.2 Structural Delay of Convolutional Codes

We focus on encoding and decoding of convolutional codes as described in Section 5.2 and restrict our investigations to codes of rate 1/2. As can be deducted from Section 5.2, encoded symbols are readily available as soon as the first source symbol enters the encoder. As the output needs to transport $1/R$ encoded symbols and again $T_{\mathrm{s}} = R \cdot T_{\mathrm{b}}$ is chosen, the encoded symbols can be transported over the channel immediately. Concluding, encoding and transmission of convolutional codes does not impose a structural delay. However, a delay has to be accepted for decoding, as the decoder needs access to past and future received symbols to make use of the code rules. Access to the received symbols transmitted later than the one that is currently to decode causes structural delay. The values w_{p} and $w_{\boldsymbol{\beta}}$, introduced in Section 5.2.2, specify this look-ahead exactly and specify the structural delay as multiples of T_{b}. In our upcoming discussions, we treat w_{p} and $w_{\boldsymbol{\beta}}$ as user parameters and assign reasonable values to them.

8.2 Performance Results: Comparison of Convolutional Codes and LDPC Codes

In this section, we compare block codes and convolutional codes, searching for the best compromise of low structural delay and good decoding performance. To this end, we first discuss the performance results for the different codes and subsequently show the SNR ($10 \cdot \log_{10}(E_{\mathrm{b}}/N_0)$) required for a certain reliability over the structural delay.

We are interested in performance results of convolutional codes which cover a wide range of delay values. To this end we use a multitude of codes and wide-spread parameters $w_{\boldsymbol{\beta}}$ and w_{p}. In more detail, we choose these lengths as multiples of the constraint length of non-recursive codes, $\varphi \cdot (\mathrm{ld}(|\mathcal{S}|) + 1)$, $1 \leq \varphi \leq 5$, $\varphi \in \mathbb{R}$. The resulting structural delay reads $\Delta_{\mathrm{cc}} = \varphi \cdot (\mathrm{ld}(|\mathcal{S}|) + 1)$, $1 \leq \varphi \leq 5$, $\varphi \in \mathbb{R}$. The decoding performance cannot be improved significantly by choosing $\varphi > 5$ [HJ71], and $\varphi = 0$ achieves the performance of uncoded transmission.

Figure 8.4 shows BER performance results for the BCJR and Viterbi decoder and all convolutional codes taken into consideration, cf. Section 5.2. We choose the boundary values for φ, i.e. $\varphi = 1$ and $\varphi = 5$. We observe that the decoding performance depends on the code as well as the choice of φ. The BCJR decoder obtains a slightly better performance.

In a next step, we define a reliability measure, e.g. a target BER and a target FER, respectively, and determine the required $10 \cdot \log_{10}(E_{\mathrm{b}}/N_0)$ to obtain this value. In order to classify the

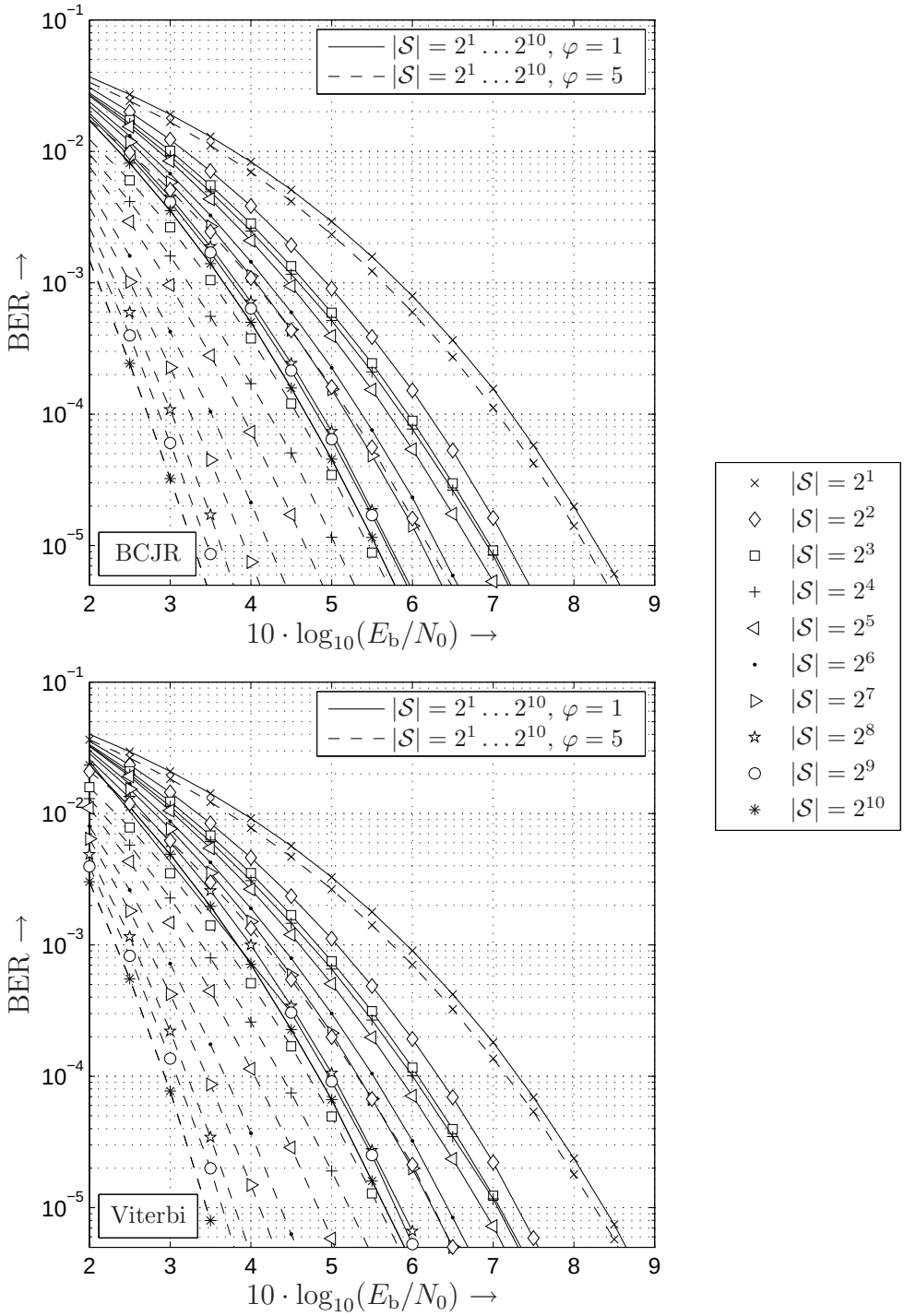


Figure 8.4: BER for rate-1/2 convolutional codes with $|\mathcal{S}| = 2^1$ to $|\mathcal{S}| = 2^{10}$ states. We set $\varphi = 1$ and $\varphi = 5$, respectively.

different convolutional codes with respect to decoding performance and structural delay, we plot the required SNR in order to obtain the target error rates over the structural delay. Figure 8.5 shows the results for the BCJR decoder, while Figure 8.6 shows these results for the Viterbi algorithm. In order to provide results on the FER when using convolutional codes, one needs to set the frame length. We define $L_\mathrm{F} = \Delta_\mathrm{cc} + 1$. This choice considers the current received symbol and the received symbols required for state vector and path estimation as one frame.

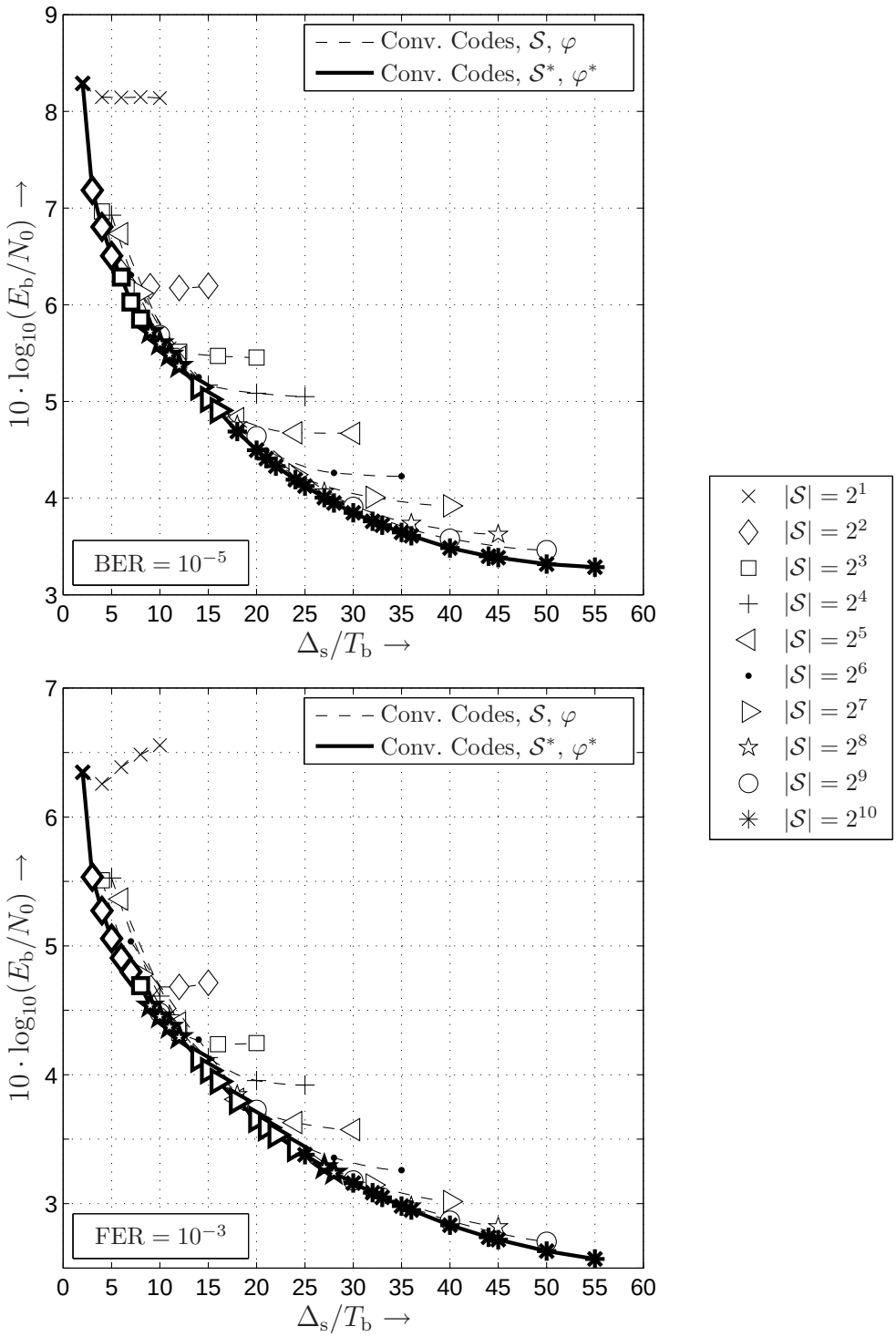


Figure 8.5: Required SNR for BER $= 10^{-5}$ and FER $= 10^{-3}$, respectively, for rate-1/2 convolutional codes with $|\mathcal{S}| = 2^1$ to $|\mathcal{S}| = 2^{10}$ states. We vary φ between $\varphi = 1$ and $\varphi = 5$. Favorable choices for a given delay are indicated by a bold marker. The BCJR decoder is used for decoding.

The insight provided by Figure 8.5 and Figure 8.6 is two-fold. First, it can be observed that the BCJR decoder performs slightly better than the Viterbi decoder, also when the FER is the performance measure of interest. The reason for this is that a sliding-window approach, cf. Section 5.2.2, is used to keep the structural delay low. In this case, the Viterbi algorithm

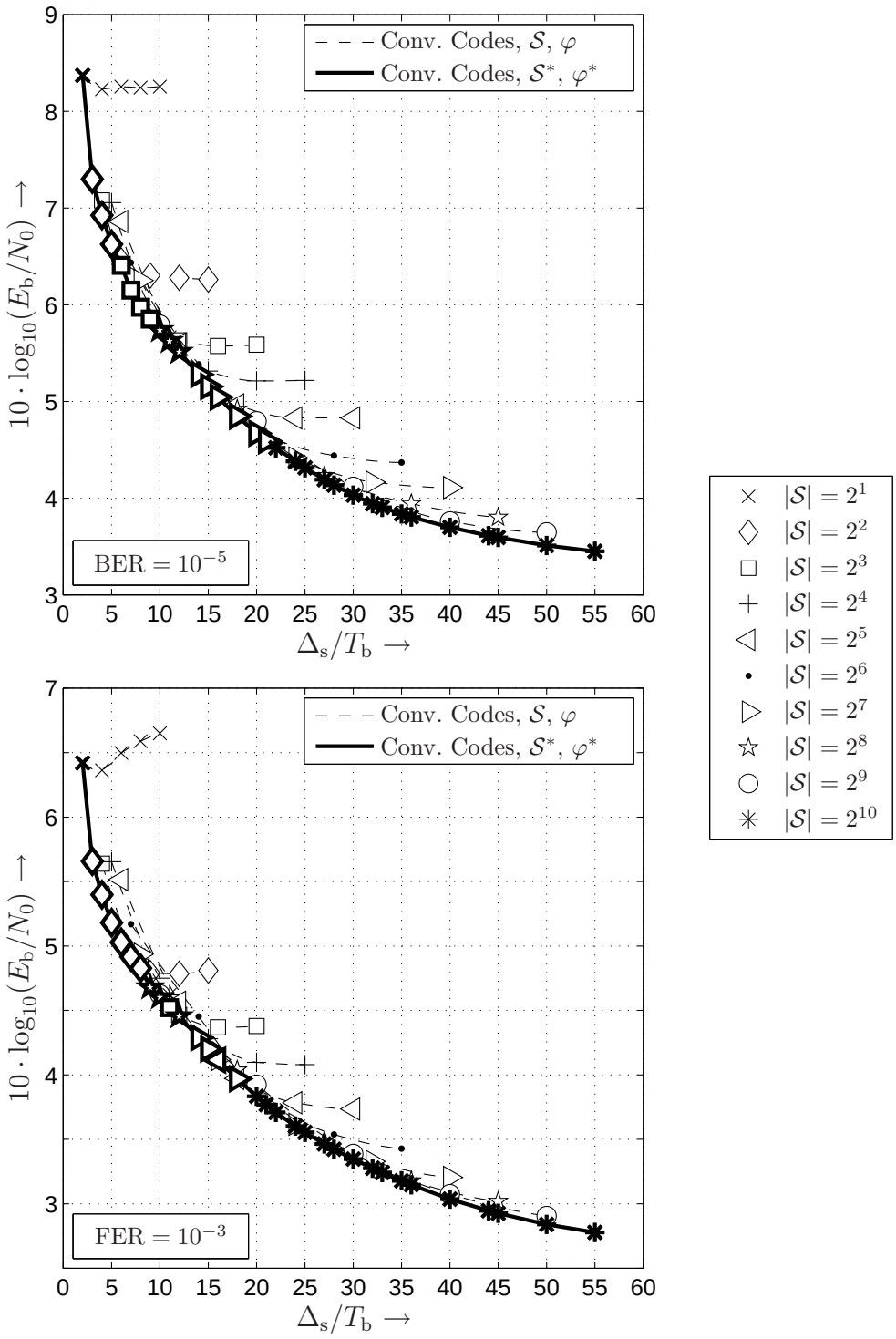


Figure 8.6: Required SNR for BER $= 10^{-5}$ and FER $= 10^{-3}$, respectively, for rate-1/2 convolutional codes with $|\mathcal{S}| = 2^1$ to $|\mathcal{S}| = 2^{10}$ states. We vary φ between $\varphi = 1$ and $\varphi = 5$. Favorable choices for a given delay are indicated by a bold marker. The Viterbi decoder is used for decoding.

cannot estimate the whole transmitted sequence, the task for which it was originally constructed [Vit67]. Instead, to keep the structural delay low, only parts of the sequence, and with this an unterminated trellis, are estimated. As can be seen, the BCJR decoder is the favorable choice for this task, independent of the target error rate.

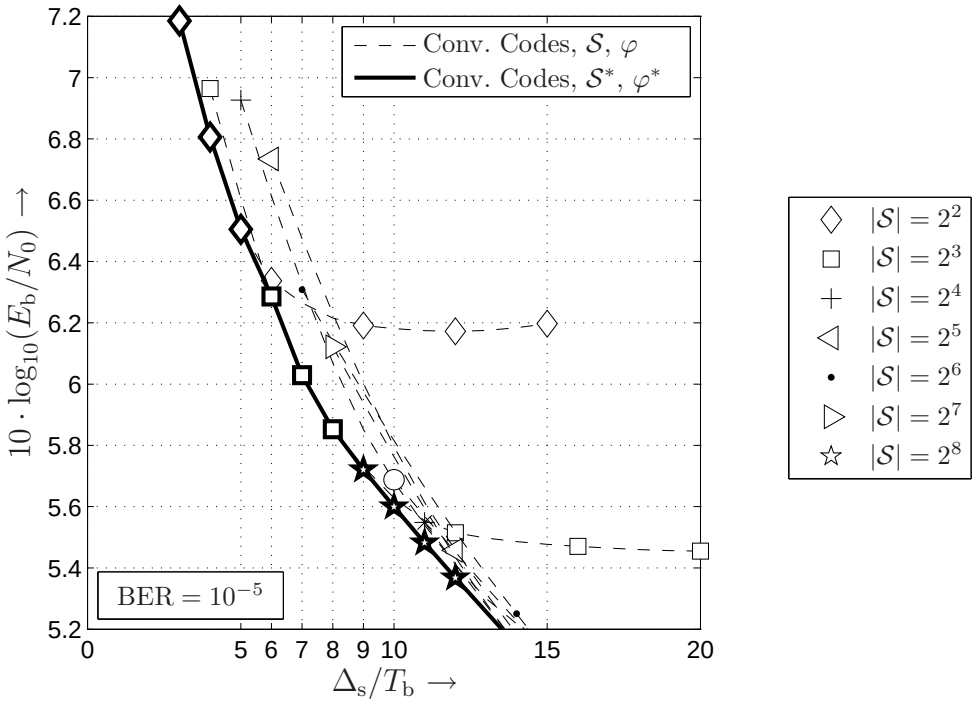


Figure 8.7: Close-up view of Figure 8.5 (BER constraint).

For a given delay, there exist favorable choices for φ and the convolutional code, denoted by $\mathcal{S}^*$ and φ^*, respectively. The bold-face markers in Figure 8.5, Figure 8.6, and Figure 8.7 indicate $\mathcal{S}^*$ and their positions mark the corresponding delay and the required SNR. The following example [HH09a] makes this coherence more clear.

Example 8.2.1 *Assume that a* BER $= 10^{-5}$ *is to be met and a delay of* 8 *time steps* T_b *can be tolerated. At the receiver, a sliding-window BCJR decoder is applied. We discuss the most reasonable choice for the convolutional code. Figure 8.5 indicates that these constraints can be met with a convolutional code with* $|\mathcal{S}| = 2^1$ *states and a choice of* $\varphi = 4$*. In this case, an SNR of at least* $10 \cdot \log_{10}(E_\mathrm{b}/N_0) = 8.15\,\mathrm{dB}$ *would be required. Another choice to obtain* $\Delta_\mathrm{cc} = 8$ *would be a* 2^4*-state code. Here, the delay is not a multiple of the constraint length; to be precise* $\varphi = 8/5$ *has to be chosen to obtain* $w_{\boldsymbol{\beta}} = 8$ *for this code. The required SNR is significantly lower,* $10 \cdot \log_{10}(E_\mathrm{b}/N_0) = 6.10\,\mathrm{dB}$*. Confer Figure 8.5 and its close-up view in Figure 8.7 to confirm this value. Both theses choices are outperformed by another code. The bold curve in Figure 8.7 indicates that the optimal choice from the considered codes is the one with* $|\mathcal{S}| = 2^3$ *states, with* $\varphi = 2$*, and a resulting SNR of* $10 \cdot \log_{10}(E_\mathrm{b}/N_0) = 5.85\,\mathrm{dB}$*.*

Codes with a high number of states do not necessarily outperform codes with a lower number of states in this comparison. Observe that $\Delta_\mathrm{cc} = 8$ *and* BER $= 10^{-5}$ *can also be obtained using a code with* $|\mathcal{S}| = 2^7$ *states and* $\varphi = 1$*. However,* $10 \cdot \log_{10}(E_\mathrm{b}/N_0) = 6.12\,\mathrm{dB}$ *is required in that case.*

Remark 8.2.1 *Figure 8.5 and Figure 8.6 are discussed. We point out that the required SNR rises with increasing delay if* FER $= 10^{-3}$ *is to be met and a code with a low number of states is*

considered. With rising structural delay, the parameter $w_{\mathcal{B}}$ increases. This results in an improved decoding performance. However, the window length L_{F} increases as well. As a consequence, frame errors become more likely.

Let us now discuss the results for the rivaling PEG-optimized LDPC codes. These codes follow the variable nodes degree distribution $\Lambda_3(x)$, specified in Section 5.1.2. The corresponding code lengths n vary between $n = 50$ and $n = 1000$. In our simulations, the maximum number of iterations for the BP decoder is set to $I_{\max} = 200$. This choice is motivated by the fact that increasing this number further does not lead to measurable performance improvements. Recall that we are interested in good decoding performance at low structural delay and that the latter parameter is not affected by $I_{\max}$.

The results are depicted in Figure 8.8, along with the threshold-bound for $\psi_{\mathrm{L}_3(x),\mathrm{R}_3(x)}$ and the capacity bound obtained with BPSK signaling and a channel code of rate 1/2. For details on bounds of this type, we refer the reader to [HH09b].

We observe from Figure 8.8 that a significantly better BER performance is obtained if systematic encoding instead of non-systematic encoding is used. However, the FER is not affected by this choice as the post-processing step described in Section 5.1.3 does not have an impact on this error rate. For all block codes we set $L_{\mathrm{F}} = n \cdot R$, i.e. we measure frame errors on blocks of source symbols. The depicted results require further commentary. The PEG construction method is a random construction method which follows the degree distribution in a greedy manner but uses random decisions whenever the cost function cannot identify the best choice for the next step unambiguously [HEA05]. We state that all the codes within the considered set of PEG-code ensembles do depend on the initial setting of the random number generator. For the error rates of interest, the performance is yet concentrated, such that it is sufficient to show one representative example. Concentrated performance of an ensemble of codes within the waterfall region is common in the field of channel coding. Results confirming this statement were reported in [CCSV08, RU08].

In the following, we also take the structural delay of the block codes into consideration and compare them to the lower bound for convolutional codes, defined by $\mathcal{S}^*, \varphi^*$. For the block codes, we distinguish between SPNSE, SPSE, and BlP. Also included are results for the WiMAX codes of length $n = 576$, $n = 768$, $n = 864$, and $n = 960$, cf. Section 7.7.2. Figure 8.9 shows the results for BER $= 10^{-5}$. The most prominent observation is the significant gap between convolutional codes and PEG-optimized LDPC codes for a given SNR. Example 8.2.2 details this observation.

Example 8.2.2 *A target* BER *of* 10^{-5} *at* $10 \cdot \log_{10}(E_{\mathrm{b}}/N_0) = 4\,\mathrm{dB}$ *is to be obtained. For convolutional codes, one would have to accept a structural delay of* $\Delta_{\mathrm{cc}} \approx 27.0$ *and* $\Delta_{\mathrm{cc}} \approx 30.7$ *time steps of duration* T_{b} *when using the BCJR and the Viterbi algorithm, respectively. For the same task, the PEG codes impose* $\Delta_{\mathrm{bc}} \approx 210.5$ *time steps of duration* T_{b} *for serial processing with non-systematic encoding,* $\Delta_{\mathrm{bc}} \approx 117.5$ *time steps for serial processing with systematic encoding, and* $\Delta_{\mathrm{bc}} \approx 78.3$ *time steps of duration* T_{b} *if block processing is applied.*

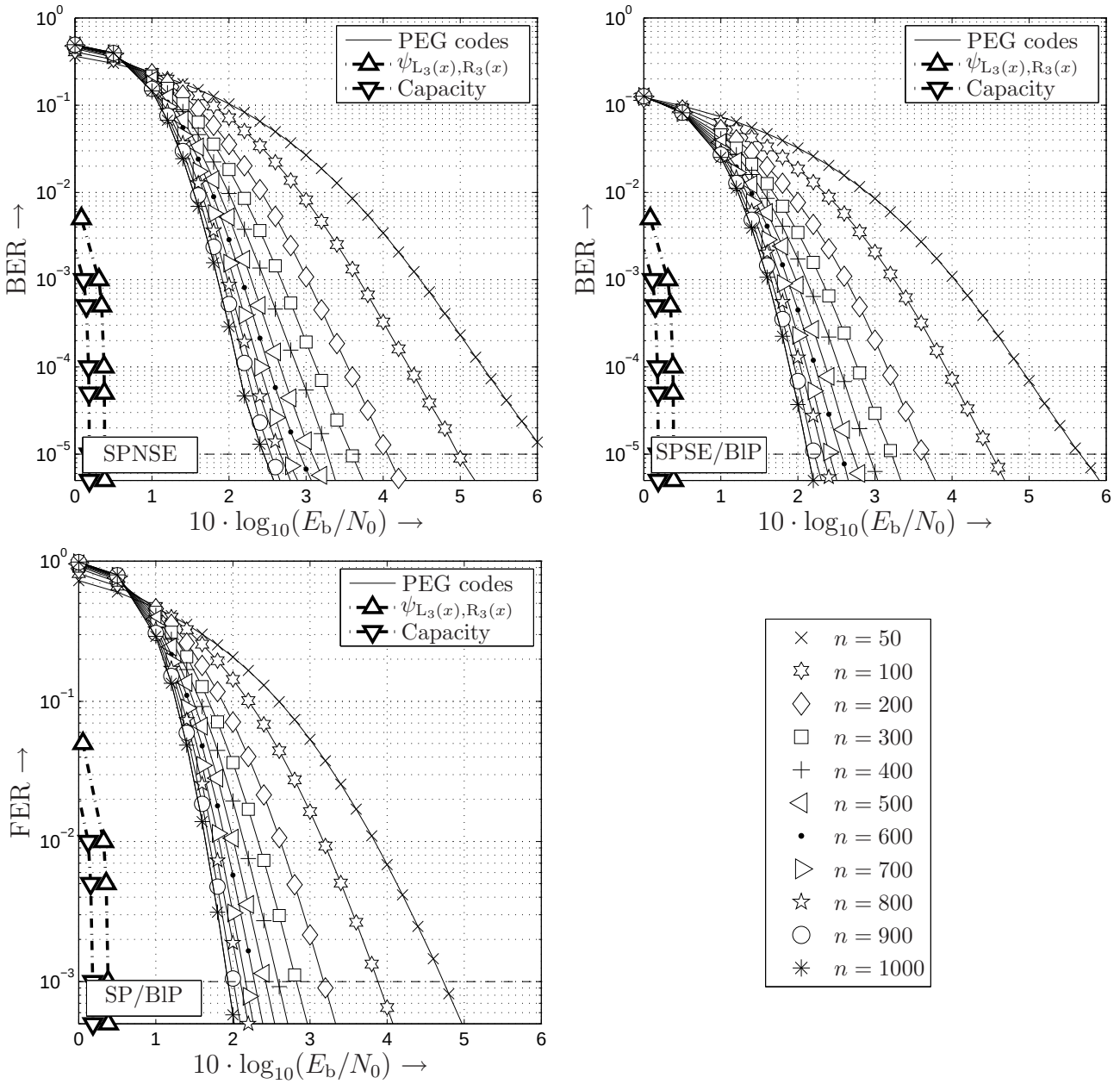


Figure 8.8: Performance curves of the PEG-optimized block codes used in this thesis. For the BER, we distinguish between SPSNSE on the one hand, and SPSE or BlP on the other hand. Contrary, no differentiation is required for the FER, as all processing approaches obtain equal frame error rates.

The WiMAX codes show a different error-floor behavior for the different code lengths, in particular the code of length $n = 864$. This explains that the results for these codes are slightly unsteady, cf. Figure 8.9. Similar observations are made for all other signal-to-noise ratios and also for the case that a target FER is specified, cf. Figure 8.10. A further important result is the fact that LDPC codes do outperform convolutional codes significantly in terms of error-prevention performance. If the structural delay is an important factor, convolutional codes are the first choice.

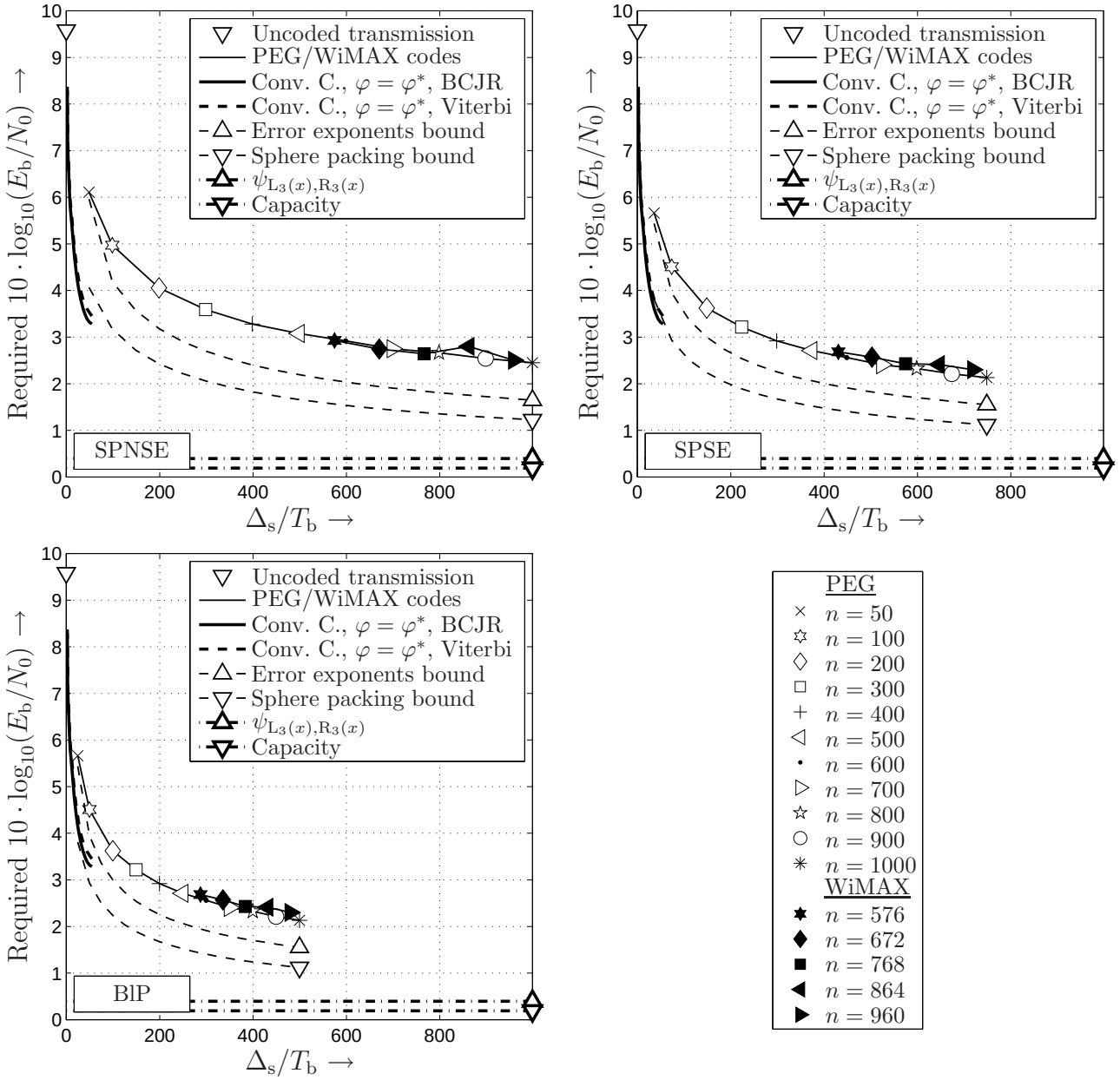


Figure 8.9: Required SNR to obtain BER = 10^{-5} using convolutional codes, PEG codes, and WiMAX codes. We distinguish between SPNSE, SPSE, and BlP.

In the following, we relate the performance results of these two classes of codes to theoretical bounds for block codes, cf. Section 5.1.4, by means of the structural delay. At first, it is surprising that the convolutional codes outperform the Gallager bound (random-coding bound) and are even located left of the sphere packing bound when serial processing is chosen. Possible explanations for this behavior include the comparison over the structural delay and the fact that convolutional codes do not obey a block structure. Consequently, the sphere packing bound does not mark a non-existence bound for them. We deduct from this comparison that block codes of rate 1/2 cannot outperform convolutional codes of the same rate when serial processing is applied and both the decoding performance and the structural delay are taken into account.

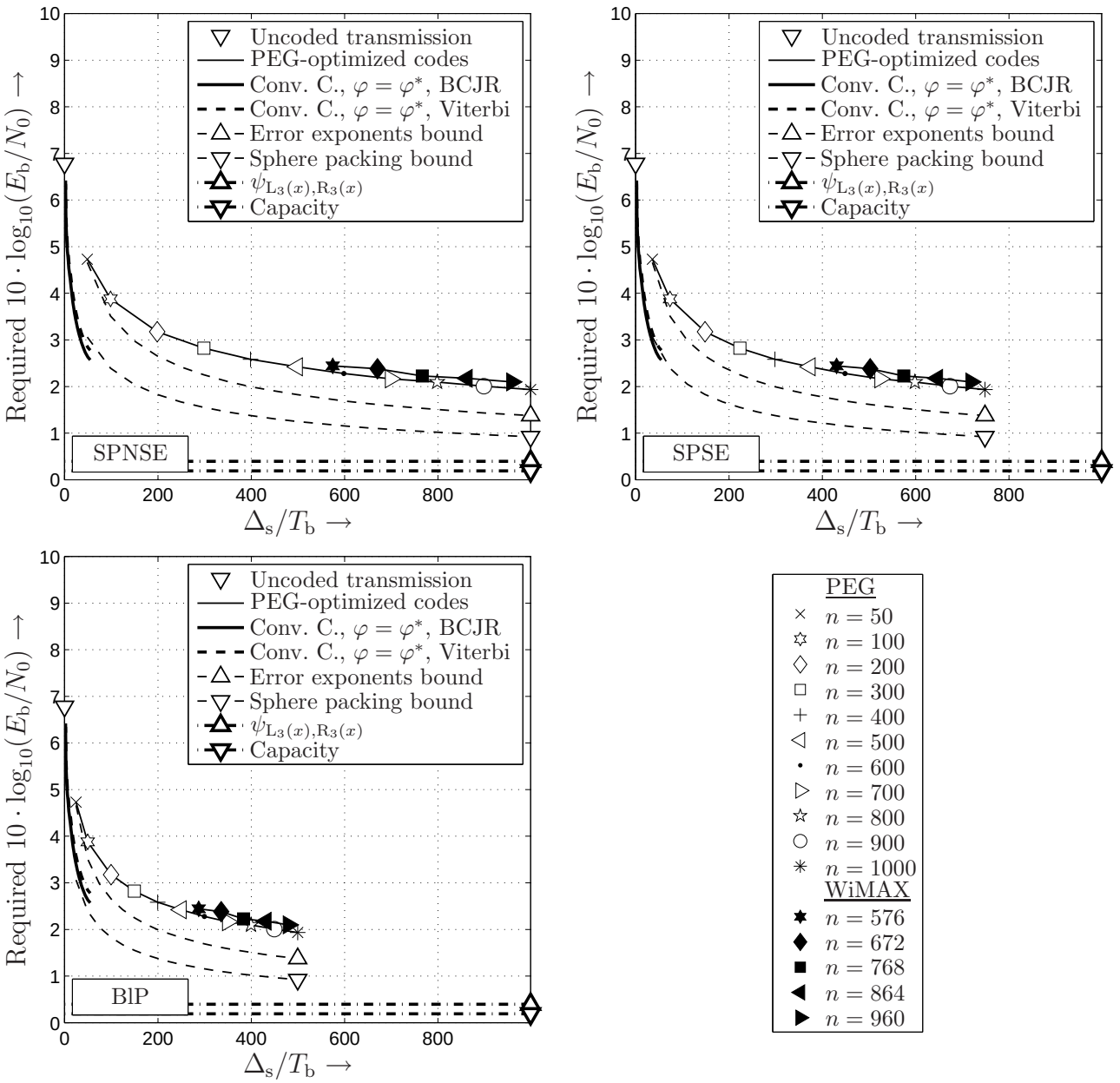


Figure 8.10: Required SNR to obtain FER $= 10^{-3}$ using convolutional codes, PEG codes, and WiMAX codes. We distinguish between SPNSE, SPSE, and BlP.

If only the decoding performance is of interest, the PEG-optimized LDPC codes obtain very remarkable results. They approach the Gallager bound very closely for short lengths. For moderate lengths of $200 \leq n \leq 1000$ and serial processing with systematic encoding, an almost constant gap of only 0.6 dB to the Gallager bound exists. The sphere packing bound is achieved by about 1.85 dB for $n = 50$ and around 1 dB for $n = 1000$. Comparable performance results are also obtained by the WiMAX-LDPC codes of length $n \leq 1000$. Similar observations are made when the FER is used as the performance criterion, cf. Figure 8.10.

In the following, we improve the performance of the block codes, i.e. the PEG and WiMAX-LDPC codes by using the L-MBBP approach. As L-MBBP allows for a completely parallel

decoding setup, we obtain an improved performance without increasing the structural delay. We choose to restrict our attention to SPSE in order to keep the presentation of the results clearly arranged. Figure 8.11 shows the results for PEG and WiMAX codes obtained with the L-MBBP setup. A close-up view of the performance of the PEG codes and WiMAX codes is provided in Figure 8.12. Results using convolutional codes are depicted for comparison reasons. Table 8.1 and Table 8.2 list the SNR required when using BP and L-MBBP decoding. In this context, the leaking algorithm in its probability-based approach with $p_{\mathcal{L}} = 0.9$ is applied. Due to the high number of representations for WiMAX codes, we limit l_{MBBP} to $2 \cdot 15$ for this class of codes. For the class of PEG codes only few realizations exist. Hence we choose to apply all available representations. The maximum number of decoders is applied for the code of length 100, where $l_{\text{MBBP}} = 2 \cdot 31$ could be set. Using this approach, the SNR required for obtaining the target error rate can be lowered by up to 0.5 dB. According to Table 8.1 and Table 8.2 at least 0.08 dB can be gained from this approach for all types and lengths of codes considered. We emphasize that using L-MBBP and optimized PEG codes, the Gallager bound can be approached by 0.5 dB or less within $n \leq 1000$.

		BER $= 10^{-5}$		FER $= 10^{-3}$	
Code	l_{MBBP}	BP	L-MBBP	BP	L-MBBP
$n = 50$	$2 \cdot 19$	5.66 dB	5.09 dB	4.73 dB	4.30 dB
$n = 100$	$2 \cdot 31$	4.51 dB	4.11 dB	3.88 dB	3.49 dB
$n = 200$	$2 \cdot 26$	3.62 dB	3.26 dB	3.18 dB	2.87 dB
$n = 300$	$2 \cdot 30$	3.22 dB	2.89 dB	2.82 dB	2.58 dB
$n = 400$	$2 \cdot 22$	2.92 dB	2.65 dB	2.58 dB	2.39 dB
$n = 500$	$2 \cdot 13$	2.71 dB	2.58 dB	2.42 dB	2.27 dB
$n = 600$	$2 \cdot 13$	2.56 dB	2.37 dB	2.28 dB	2.11 dB
$n = 700$	$2 \cdot 11$	2.41 dB	2.28 dB	2.16 dB	2.04 dB
$n = 800$	$2 \cdot 11$	2.32 dB	2.22 dB	2.10 dB	1.96 dB
$n = 900$	$2 \cdot 10$	2.21 dB	2.12 dB	2.01 dB	1.88 dB
$n = 1000$	$2 \cdot 10$	2.13 dB	2.05 dB	1.94 dB	1.83 dB

Table 8.1: Required SNR ($10 \cdot \log_{10}(E_{\text{b}}/N_0)$) to obtain a given reliability measure for PEG codes from the set of PEG-code ensembles $\mathcal{U}_3$.

		BER=10^{-5}		FER=10^{-3}	
Code	l_{MBBP}	BP	L-MBBP	BP	L-MBBP
$n = 576$	$2 \cdot 15$	2.68 dB	2.56 dB	2.44 dB	2.29 dB
$n = 672$	$2 \cdot 15$	2.57 dB	2.47 dB	2.38 dB	2.19 dB
$n = 768$	$2 \cdot 15$	2.43 dB	2.30 dB	2.23 dB	2.10 dB
$n = 864$	$2 \cdot 15$	2.41 dB	2.28 dB	2.18 dB	2.05 dB
$n = 960$	$2 \cdot 15$	2.30 dB	2.22 dB	2.10 dB	1.96 dB

Table 8.2: Required SNR ($10 \cdot \log_{10}(E_{\text{b}}/N_0)$) to obtain a given reliability measure for codes applied in the WiMAX standard of rate 1/2 and various lengths.

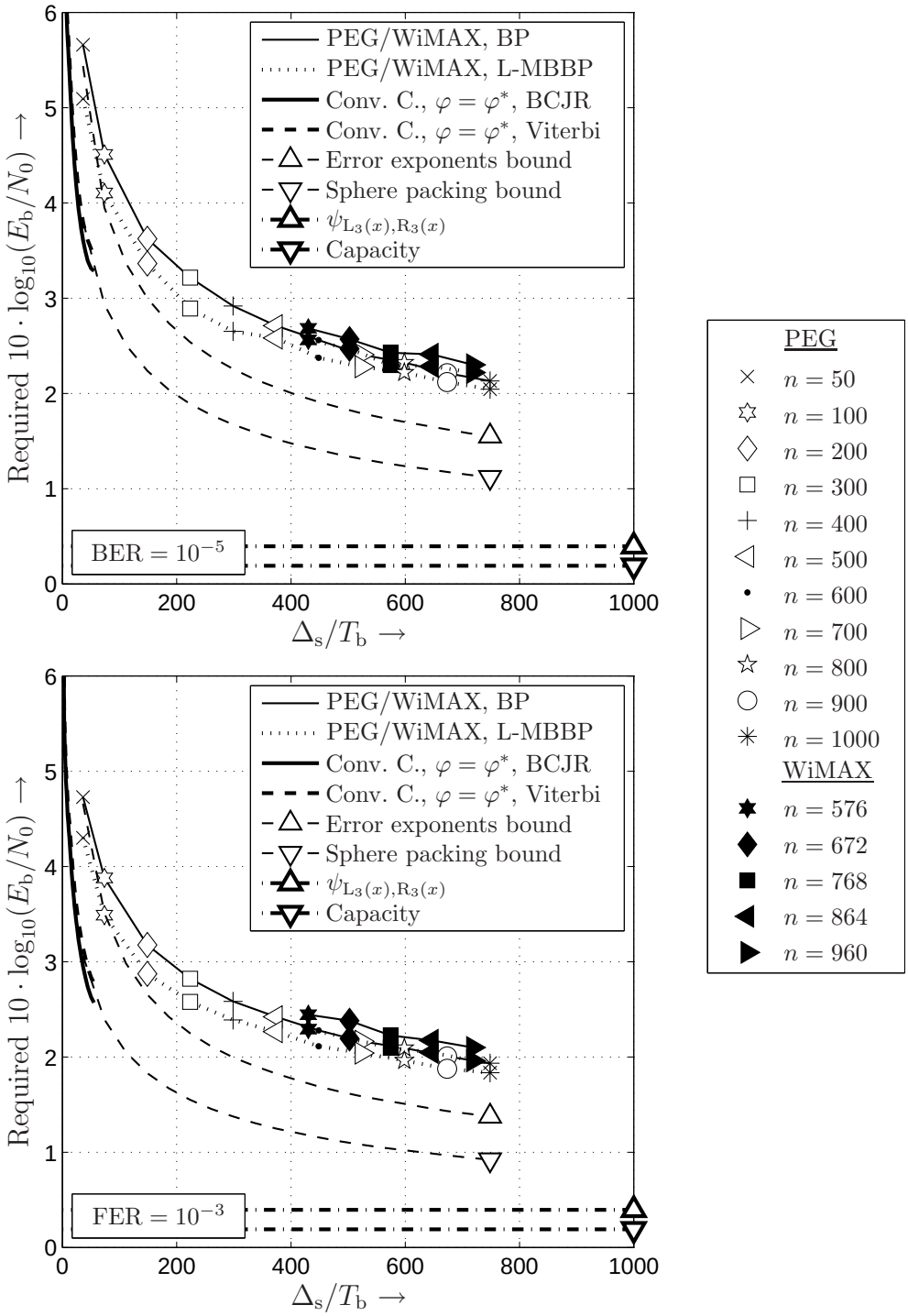


Figure 8.11: Required SNR to obtain BER $= 10^{-5}$ and FER $= 10^{-3}$, respectively. Results for BP decoding and MBBP decoding are given.

The performance improvement by L-MBBP mitigates the gap between convolutional codes and the PEG codes. The convolutional codes obtain the target error rates with the lowest-possible SNR at a structural delay of 55 time steps T_b. PEG codes using BP impose a delay of 210.7 time steps T_b to reach BER $= 10^{-5}$ at the same SNR, which reads 3.29 dB, cf. Figure 8.11 and Figure 8.12. With L-MBBP, this value decreases significantly to 160.9 time steps T_b. Similar results exist for the target FER of 10^{-3}, where 2.57 dB are required. The original gap lies between

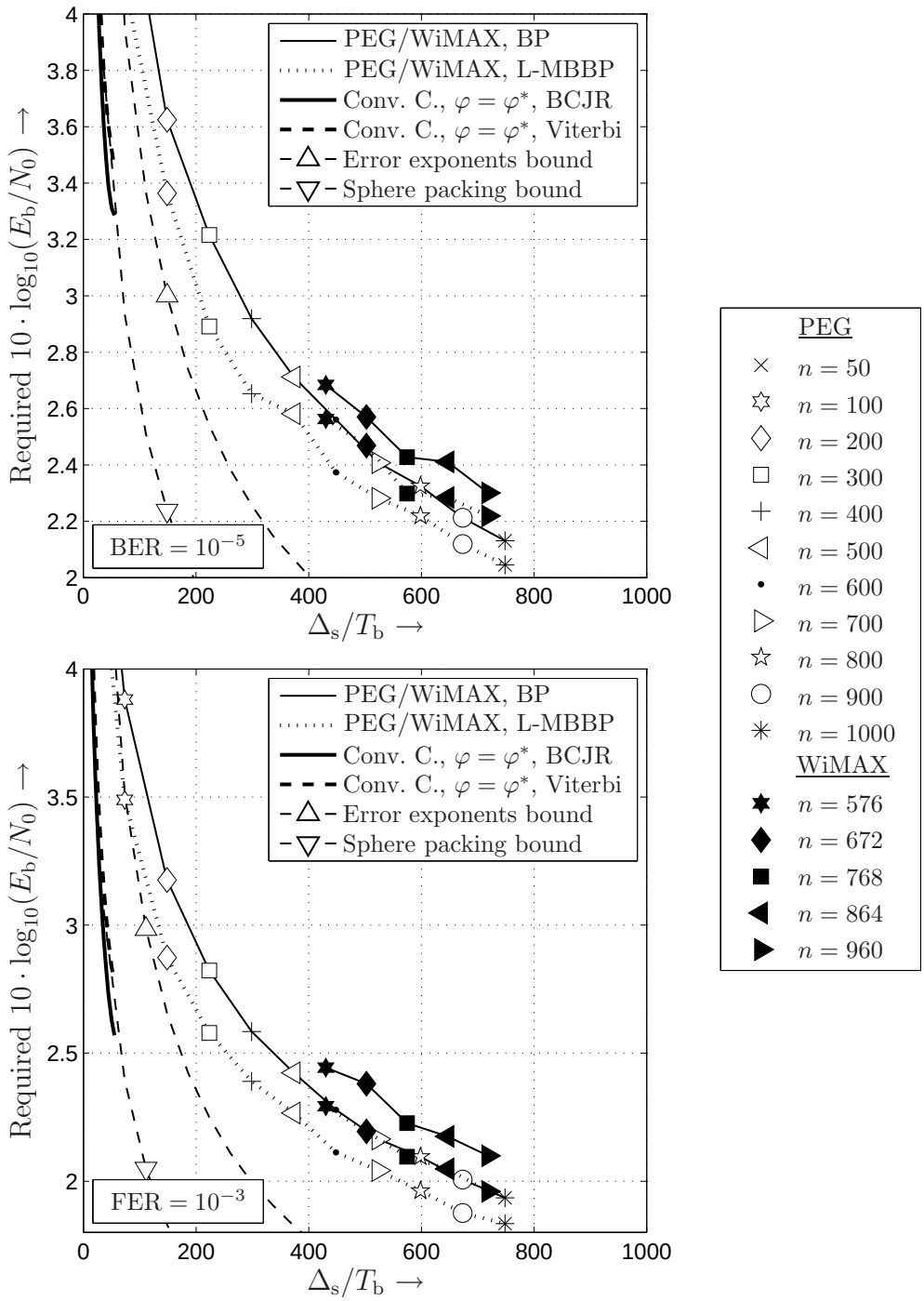


Figure 8.12: Required SNR to obtain BER $= 10^{-5}$ and FER $= 10^{-3}$, respectively. A close-up view of the results from Figure 8.11 with focus on PEG codes and WiMAX codes is given.

55 and 305.3 time steps T_b and is mitigated to a gap between 55 and 227.1 time steps T_b. Results using L-MBBP for selected algebraic codes of different lengths and rate close to 1/2 are available, cf. Appendix D.

8.3 Decoding Performance and Total Delay

We have observed that convolutional codes are a good choice for applications with low structural delay. In this section, we widen our focus and classify coding schemes with respect to decoding performance and the total delay. We denote the processing delay as Δ_p and the total delay as $\Delta_\mathrm{t} = \Delta_\mathrm{s} + \Delta_\mathrm{p}$. Note that the processing delay includes both the delay for encoding and decoding. For both convolutional codes and LDPC codes, the encoding does not take a significant share, but is still respected in our results. The processing delay due to the mapping function is neglected.

In order to determine Δ_p, we find the processing time by simulation and express it as multiples of T_b. In the simulation setup, we apply quick implementations in C for the encoding and decoding units running on a standard Linux computer[2]. In order to find a valid value for T_b, we consider the transmission rates of the WiMAX standard. The reason for this choice is that WiMAX is a popular standard and that the rate and lengths of the codes match the parameters we are interested in. Using QPSK modulation and different frequency bandwidths, data rates between 1.04 MBit/s and 16.62 MBit/s can be obtained [JY04]. This corresponds to 0.52 MBit/s and 8.31 MBit/s per real dimension. We calculate the corresponding bit duration as a value between $T_\mathrm{b} \approx 1.92 \cdot 10^{-6}$ s and $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}$ s. These two boundaries are used in our subsequent investigations. In order to relate the processing delay and the structural delay, we express the measured processing delay as multiples of T_b.

In a first step, we restrict our attention to convolutional codes. When using the BCJR or Viterbi decoder, the processing delay of this class of codes does not depend on the current signal to noise ratio. Recall that the structural delay for these decoders also does not depend on the signal-to-noise ratio. Instead, it is solely specified by the window lengths for backward estimation, w_p and $w_{\boldsymbol{\beta}}$, respectively (cf. Section 5.2.2). These values are specified by the choice of the code and the additional parameter φ, cf. Section 8.2.

In Figure 8.13 we observe that the BCJR decoder requires a significantly higher processing complexity compared to the Viterbi decoder. It is worth mentioning that the processing delay of the Viterbi decoder is constant over the structural delay while the processing delay of the BCJR decoder rises. This is a consequence of the fact that the BCJR decoder requires a backward recursion while the Viterbi decoder does not. Both decoders can perform the forward recursion up to the symbol received most recently. Whenever a new symbol is received, the additional step in the forward recursion can be done in an incremental manner. In order to estimate the current source symbol, the Viterbi decoder just needs to trace the most-probable path in the state transition diagram. Contrary, the sliding-window version of the BCJR decoder (cf. Section 5.2.2), has to perform a complete backward recursion in this diagram. This is less complex than a standard BCJR decoder but still more complex than the Viterbi algorithm. As this step has to be performed whenever a new symbol is received, the complexity of this decoder strongly depends on the length of this estimation interval, i.e. the structural delay. Straightforward, the

[2] AMD Athlon 64 3200+ (single core with 2.2 GHz), 2 GB RAM, 2.6.11 series Linux kernel.

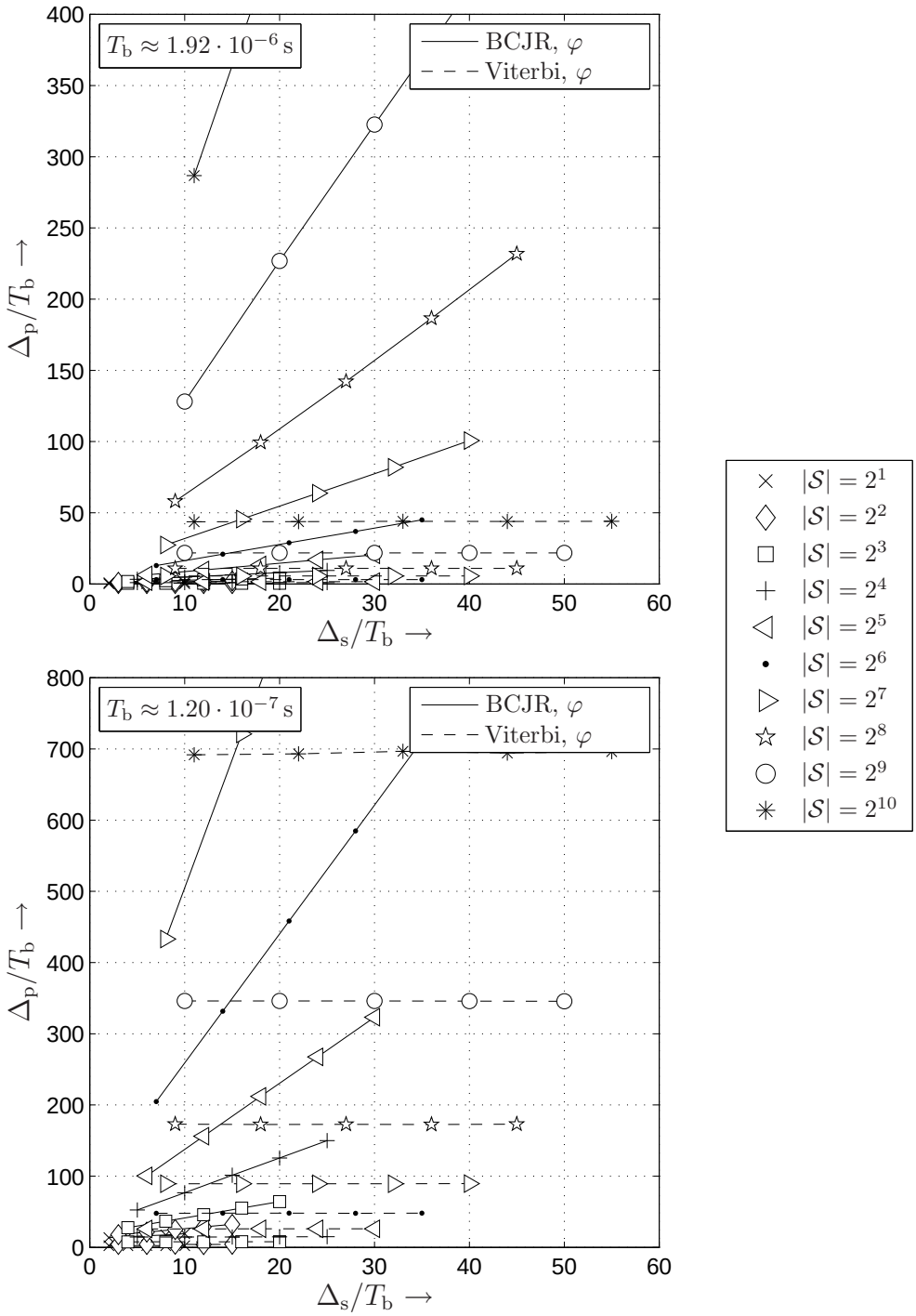


Figure 8.13: Processing delay over structural delay for minimum and maximum data rate of the WiMAX standard. Both the Viterbi and the BCJR decoder are used for decoding. Bit duration $T_\mathrm{b} \approx 1.92 \cdot 10^{-6}\,\mathrm{s}$ corresponds to a data rate of 0.52 Mbit/s and $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$ corresponds to a data rate of 8.31 Mbit/s.

inclination of the curves also depends on the number of states. The Viterbi decoder is up to a factor of 27 faster than the BCJR decoder. Figure 8.13 also indicates that the processing delay of the Viterbi decoder is comparable to the structural delay of the block codes, cf. Figure 8.11. Henceforth, we restrict our attention to the Viterbi decoder, despite of its minor performance loss compared to the BCJR decoding algorithm.

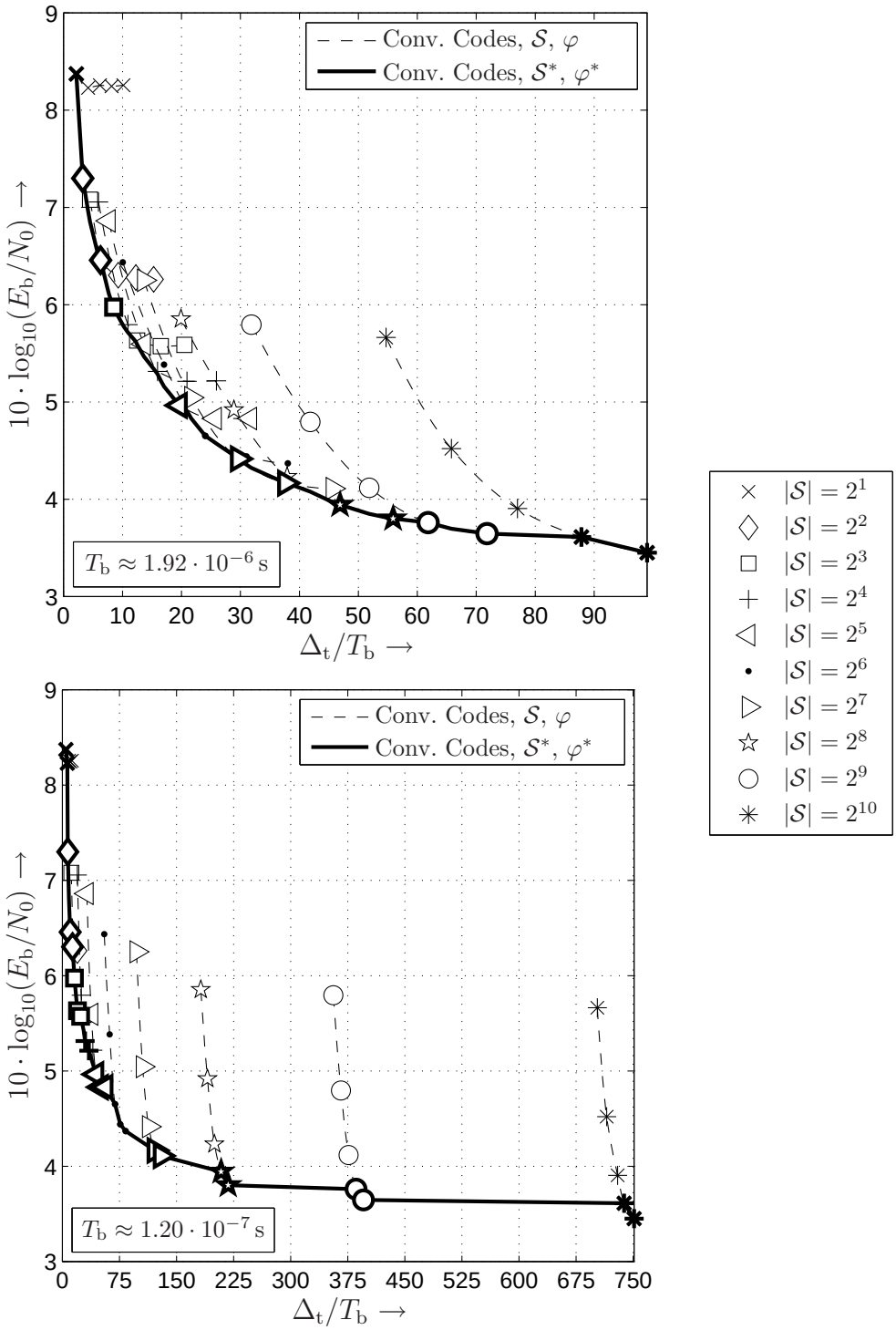


Figure 8.14: Required SNR for BER $= 10^{-5}$ using rate-1/2 convolutional codes with $|\mathcal{S}| = 2^1$ to $|\mathcal{S}| = 2^{10}$ states and $\varphi = 1$ to $\varphi = 5$, respectively. The Viterbi decoder is used for decoding. Bit duration $T_\mathrm{b} \approx 1.92 \cdot 10^{-6}\,\mathrm{s}$ corresponds to 0.52 Mbit/s and $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$ corresponds to 8.31 Mbit/s. Due to their multiplicity, only a subset of the bold-face markers is shown.

In Figure 8.14 we show the SNR required to meet the target error rate BER $= 10^{-5}$ over the total delay Δ_t for the case that convolutional codes are used. Alike for the structural delay, the bold curve marks a favorable choice for each total delay. The shape of the resulting curve is

similar to the curve over structural delay. This holds in particular for transmission at low data rates.

In a second step, we discuss the processing delay of LDPC codes, decoded with the BP implementation following [Nea]. Here, we distinguish between the average processing delay and the maximum processing delay. The average delay is the processing delay obtained on the average when at most $I_{\max}$ iterations are allowed. Here, the decoding process is stopped as soon as a valid codeword is found. The maximum delay is obtained when exactly $I_{\max}$ iterations are performed. We show the average processing delay in order to obtain BER $= 10^{-5}$ as well as the maximum processing delay, which does not depend on the SNR, over the structural delay in Figure 8.15. Although a practical BP decoder is always built in such a way that it stops decoding as soon as its convergence is observed, the investigation on the maximum delay makes sense. A delay-critical application is usually designed to work valid for the worst-case scenario, i.e. the case when the decoder uses all iterations available. This type of processing delay is henceforth considered as the processing delay for block codes. Figure 8.15 shows the processing delay for PEG-optimized codes and the codes from the WiMAX standard as a function of the structural delay.

The following observations are worth mentioning. In the upper part of the figure the maximum processing delay is shown, while the lower curves and markers show the processing delay when the decoders are allowed to stop as soon as they converge. For the latter case, we distinguish between the constraints BER $= 10^{-5}$ and FER $= 10^{-3}$, but report that the results are comparable. The maximum processing delay for the fastest transmission speed in the WiMAX standard is still lower than the structural delay of these coding schemes. Furthermore the upper bound on the maximum processing delay shows that WiMAX codes can be decoded faster than PEG codes. This originates from the fact that the density of the parity-check matrix for the WiMAX codes is lower than the density of the corresponding PEG code. Relating the density of the WiMAX code of length 960 to the PEG code of length 1000 results to a factor of 0.86. This value corresponds well to the reduction of processing delay.

With the structural delay and processing delay readily available, we compare the decoding performance of convolutional codes and block codes with respect to decoding performance and the total delay. The results for $T_{\mathrm{b}} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$ are given in Figure 8.16. These results confirm the conclusions based on the comparison over structural delay. If the total delay is limited, convolutional codes are the first choice. If higher values are allowed for the processing delay, the discussed block codes obtain given error rates at lower SNRs and should hence be preferred. In our setup, convolutional codes are the first choice up to a delay of 538.8 time steps T_{b} for BER $= 10^{-5}$ and 685.0 time steps T_{b} for FER $= 10^{-3}$, respectively.

As a last step, we compare the performance of BP decoding and L-MBBP decoding over the total delay. In this investigation, the processing delay of an L-MBBP decoder and a BP decoder are assumed to be equal. This is attributed to the fact that we take the delay for exactly 200 iterations into consideration and that the L-MBBP approach is a completely parallel

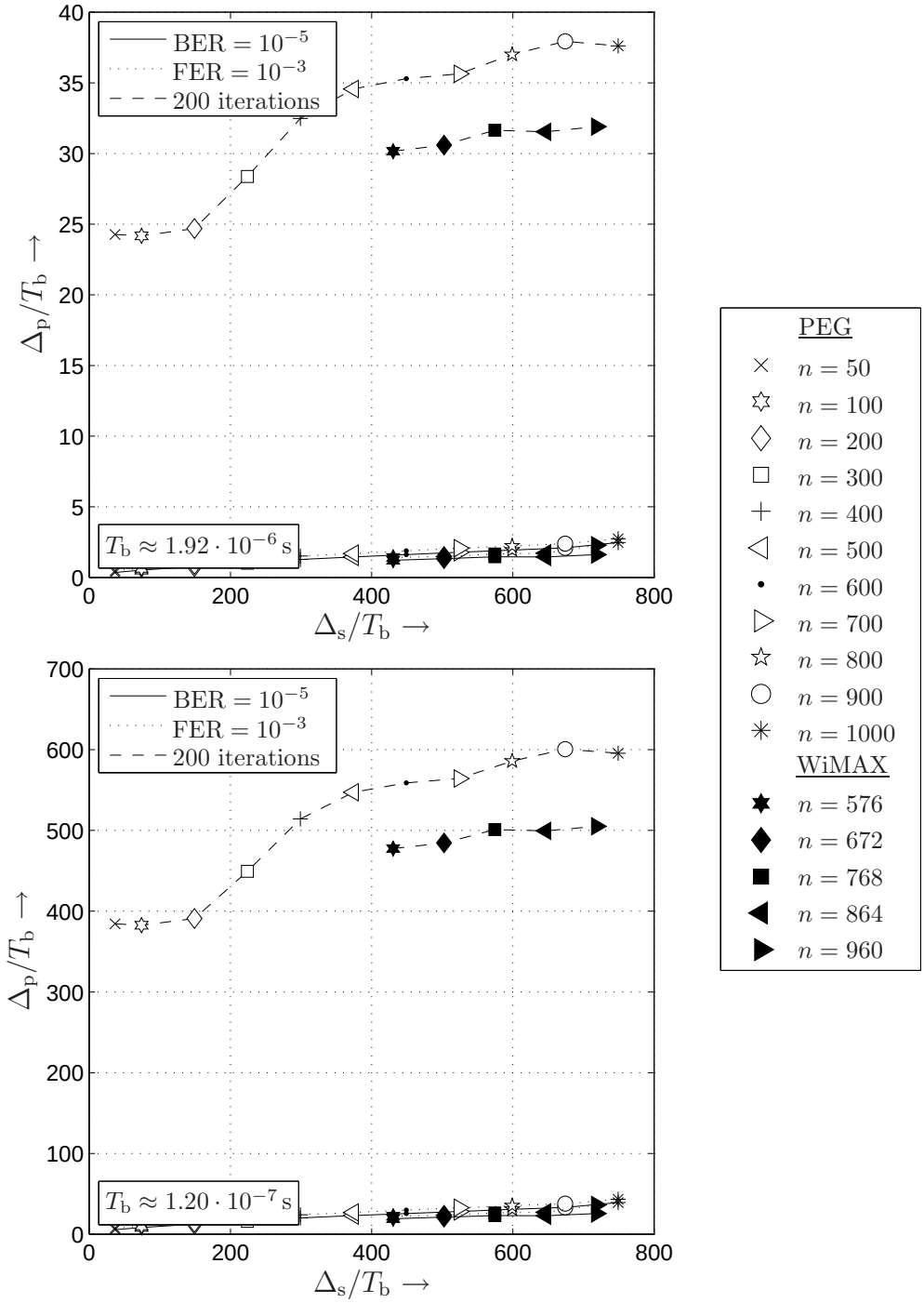


Figure 8.15: Processing delay over the structural delay. Marked are the processing delay required to obtain BER $= 10^{-5}$ with at most 200 iterations, the processing delay required to obtain FER $= 10^{-3}$ with at most 200 iterations, and the processing delay obtained by performing exactly 200 iterations. Bit duration $T_\mathrm{b} \approx 1.92 \cdot 10^{-6}\,\mathrm{s}$ corresponds to 0.52 Mbit/s and $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$ corresponds to 8.31 Mbit/s.

structure. Figure 8.17 provides these results in a close-up view. Again, the bit duration is set to $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$.

The important insight from Figure 8.16 and Figure 8.17 is the fact that the codes from the WiMAX standard can compete with the PEG-optimized codes when compared over total delay,

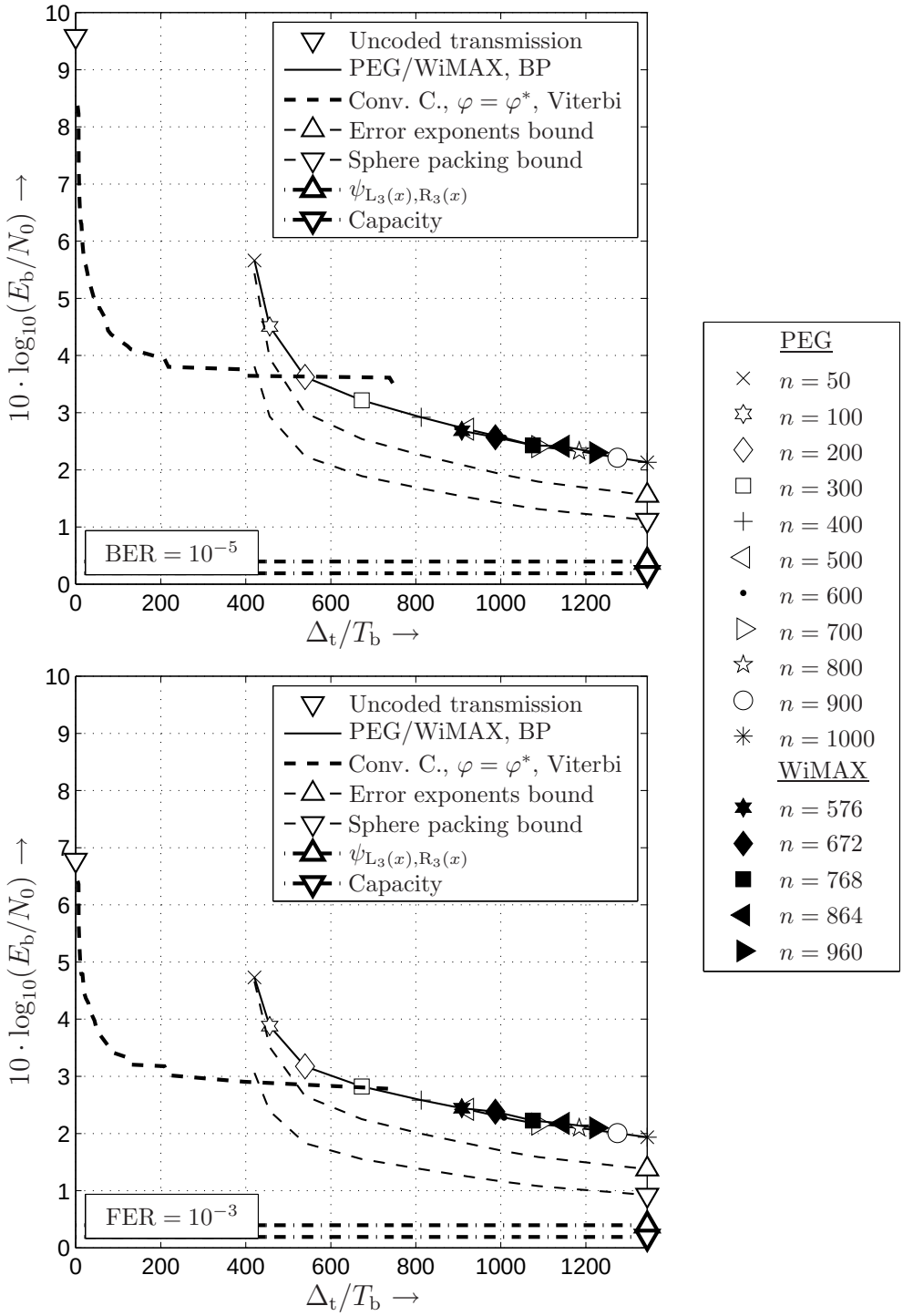


Figure 8.16: Required SNR of convolutional codes, decoded by the Viterbi decoder, as well as PEG/WiMAX codes, decoded with the BP algorithm. The bit duration $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$ corresponds to a data rate of 8.31 Mbit/s.

also if L-MBBP is applied. This can be attributed to the lower density of the parity-check matrix and the consequential lower processing delay.

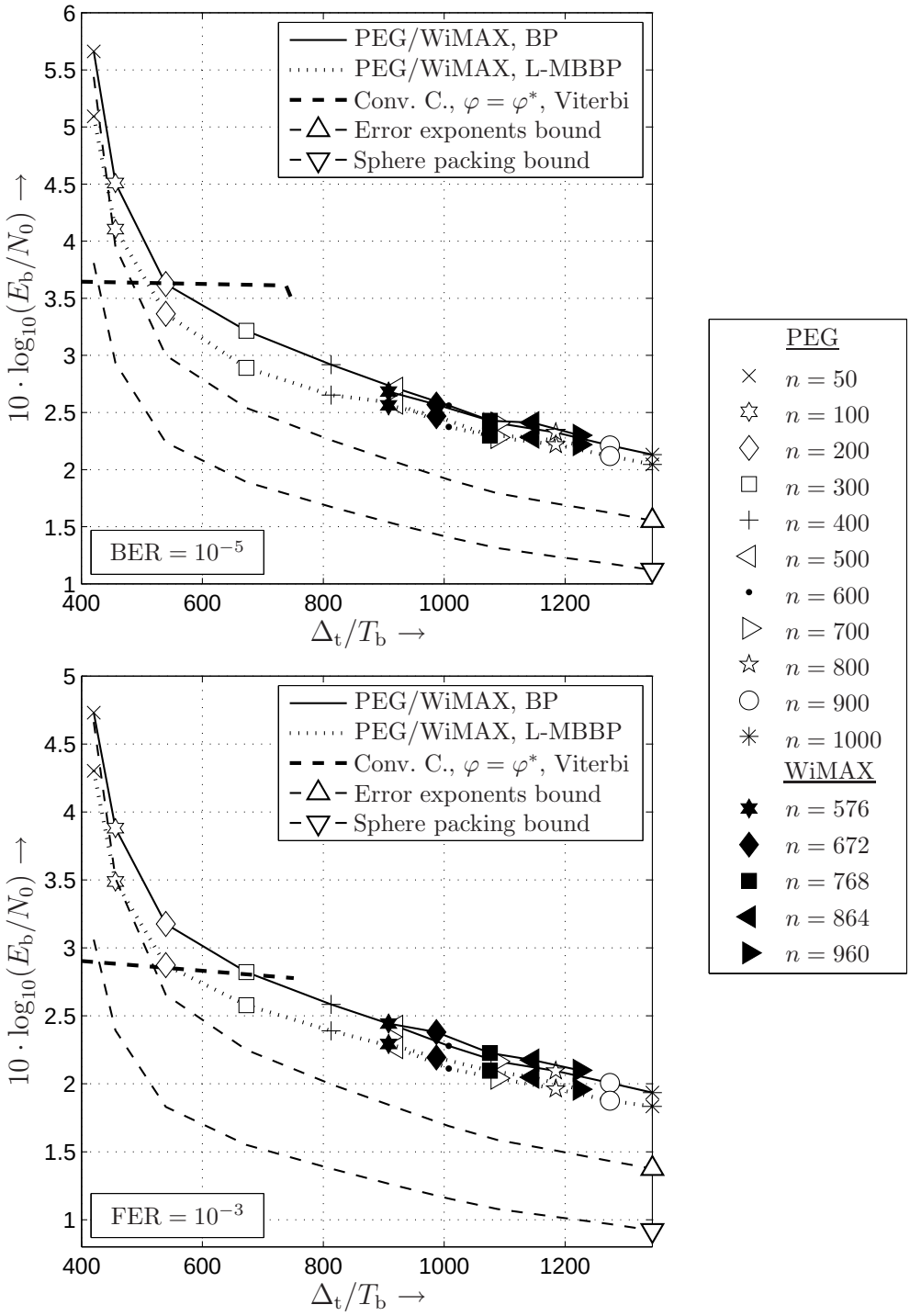


Figure 8.17: Required SNR of convolutional codes PEG/WiMAX codes with BP decoding, close-up view. The bit duration $T_\mathrm{b} \approx 1.20 \cdot 10^{-7}\,\mathrm{s}$ corresponds to a data rate of 8.31 Mbit/s.

Chapter 9

Conclusions

In this thesis, the decoding of moderate-length block codes ($n \leq 1000$) using belief-propagation decoding and redundant code descriptions was studied. It was shown that the proper use of redundant code descriptions allows for a significant performance improvement when compared to the standard approach. Subsequently, a comparison to convolutional codes with respect to decoding performance and delay was conducted. We subsume the three-fold contribution of this thesis and draw conclusions.

First, we investigated redundant belief-propagation decoding on the binary erasure channel. For this special channel model, we presented an adaptive decoding method termed stopping automorphism decoding. This scheme uses information on the code structure to generate redundant parity-check information on request. A standard belief-propagation decoder is strongly impaired by the presence of stopping sets. If such a set is encountered, the decoder has to abort the decoding process as it is not capable of recovering further erasures with its representation of the parity-check information. Contrary, the stopping automorphism decoder adaptively transforms the available information such that it can proceed with the decoding task. It was shown that this method allows for significant performance improvements for dense cyclic codes and compares favorably to several methods known from literature. In order to compare the novel approach to its standard counterpart in a theoretical manner, we introduced the stopping redundancy hierarchy.

Inspired by the success on the binary erasure channel, we presented a second decoding scheme which uses redundancy. This method, referred to as multiple-bases belief-propagation, allows for performance improvements when decoding data signaled over the additive white Gaussian noise channel. The additional challenge that comes with iterative decoding of reliability information are negative effects caused by feedback in the decoding process. These effects become more intense with increasing redundancy and usually outbalance the performance improvement obtained by redundant checks. The special partitioning of the parity information used in the multiple-bases belief-propagation setup allows to overcome these problems. Compared to standard belief-propagation decoding, significant improvements in decoding performance are obtained. Besides algebraic codes, the field of application includes low-density parity-check codes optimized with

the progressive edge-growth algorithm as well as low-density parity-check codes from the WiMAX standard. Recall that all codes are of moderate length.

As a third contribution, we asked for an optimized choice of a channel coding scheme for applications with low latency. Clearly, moderate-length block codes and convolutional codes are promising candidates for this task. The introduced multiple-bases belief-propagation approach is tailored for this application. It increases the performance while the possibility of parallelization allows to keep the latency constant. With the introduction of the structural delay we can compare convolutional codes and block codes with respect to performance and delay. This comparison is independent of implementation details and allows to identify favorable codes and schemes for different applications. A subsequent comparison also took the processing complexity into account and confirmed the practical relevance of this investigation. The results serve both as an indication for implementation on standard systems and as a lower bound on the delay in high-performance implementations of transmission schemes. Furthermore, all considered schemes were also confronted with theoretical limits from information theory. As an important result of this comparison, we state that progressive edge-growth codes in connection with the multiple-bases belief-propagation method can approach the Gallager bound by $\approx 0.5\,\mathrm{dB}$ or less. In this context, it was stated that the gain from the redundant approach lies in between $0.08\,\mathrm{dB}$ and $0.57\,\mathrm{dB}$.

Concluding, we would like to state that with rising computational power and parallel processing, the importance of both the multiple-bases belief-propagation and the structural delay becomes higher and higher. With processors using 80 or more cores [VHR+07] being developed, it is straightforward to implement multiple-bases belief-propagation as described in this thesis. Especially for communications applications which use moderate data rates but require a low latency, such as speech and video communication channels, the structural delay is about to gain attention.

Appendix A

Properties of the Normalized Multiplication

This section is based on the article [Hub02] and shows that the interval $[0,1]$ is an Abelian group with the normalized multiplication $p_1 \otimes p_2 = \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)}$ as the group operation. We show that all requirements of an Abelian group are fulfilled.

- Closure: $p_1 \otimes p_2 \in [0,1]$, if $0 \leq p_i \leq 1$, $i \in \{1,2\}$ can be assumed.

  The product $p_1 \otimes p_2$ is a continuous function and has the limit value $\lim_{p_1 \to 1} p_1 \otimes p_2 = 1$, $\forall p_2 > 0$. If p_1 tends to 1 while p_2 tends to 0, both at the same pace, $p_2 = 1 - p_1$ holds. One obtains

  $$\lim_{p_1 \to 1,\, p_2 \to 0,\, p_1 = 1-p_2} p_1 \otimes p_2 = \lim_{p_1 \to 1} p_1 \otimes (1-p_1) = \lim_{p_1 \to 1} \frac{p_1(1-p_1)}{p_1(1-p_1) + (1-p_1)p_1} = \frac{1}{2}.$$

  Also, one determines $\lim_{p_1 \to 0} p_1 \otimes p_2 = 0$, $\forall p_2 < 1$ what proofs the closure of the operation.

- Associativity: $(p_1 \otimes p_2) \otimes p_3 = p_1 \otimes (p_2 \otimes p_3)$.

$$\begin{aligned}
(p_1 \otimes p_2) \otimes p_3 &= \frac{\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} p_3}{\frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} p_3 + \frac{(1-p_1)(1-p_2)}{p_1 p_2 + (1-p_1)(1-p_2)}(1-p_3)} \\
&= \frac{p_1 p_2 p_3}{p_1 p_2 p_3 + (1-p_1)(1-p_2)(1-p_3)} \\
&= \frac{p_1 \frac{p_2 p_3}{p_2 p_3 + (1-p_2)(1-p_3)}}{p_1 \frac{p_2 p_3}{p_2 p_3 + (1-p_2)(1-p_3)} + (1-p_1)\frac{(1-p_2)(1-p_3)}{p_2 p_3 + (1-p_2)(1-p_3)}} \\
&= p_1 \otimes (p_2 \otimes p_3)
\end{aligned}$$

- Commutativity: $p_1 \otimes p_2 = p_2 \otimes p_1$.

  It is easy to see that

$$p_1 \otimes p_2 = \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} = p_2 \otimes p_1.$$

- Identity element $1/2$: $1/2 \otimes a = a \otimes 1/2 = a$.

  We recall commutativity and verify

$$p_1 \otimes 1/2 = \frac{p_1 \frac{1}{2}}{p_1 \frac{1}{2} + (1-p_1)\frac{1}{2}} = p_1.$$

- Existence of an inverse element $\bar{p}_1 = 1 - p_1$.

  Observe that

$$p_1 \otimes \bar{p}_1 = \frac{p_1(1-p_1)}{p_1(1-p_1) + (1-p_1)p_1} = 1/2.$$

Appendix B

The Automorphism Group of a Code is an Abelian Group

The automorphism group of a code $\mathcal{C}$ is a set of permutations[1] which send the code $\mathcal{C}$ into itself. We show that this set of permutations, together with the concatenation of two permutations as the group operation, forms an Abelian group. We denote the permutations by the functions $\zeta_1(\cdot)$, $\zeta_2(\cdot)$, and $\zeta_3(\cdot)$. The notation $\zeta_1(\mathcal{C})$ expresses that the permutation is applied to the codewords of $\mathcal{C}$ and $\zeta_2\zeta_1$ expresses a concatenation of permutations ζ_1 and ζ_2.

- Closure: Fulfilled per definition, as a permutation sends each codeword into another valid codeword and thus $\mathcal{C}$ into itself.
- Associativity: Straightforward, as the result of each operation in the set is the input element $\mathcal{C}$, independent of the execution order. In other words, $\zeta_1(\zeta_2\zeta_3(\mathcal{C})) = \zeta_1\zeta_2(\zeta_3(\mathcal{C}))$ holds.
- Commutativity: Fulfilled as input and output of the operation are equal, $\zeta_1(\zeta_2(\mathcal{C})) = \zeta_2(\zeta_1(\mathcal{C})) = \mathcal{C}$.
- Existence of an inverse element: Each permutation is invertible. This can easily be proven by writing the permutation as a matrix operation. As this matrix has exactly one element per row and column, we conclude that it has full rank.
- Existence of an identity element: The identity permutation exists for sure as sending each codeword into itself represents a special case of sending $\mathcal{C}$ into itself.

[1] A general automorphism group contains functions. Contrary, the automorphism group of a code is restricted to permutations.

Appendix C

The Stopping Redundancy of Cyclic Codes

We derive lower and upper bounds for cyclic codes and restrict our attention to parity-check matrices of cyclic form. The results in Section 6.2.1 are repeated and specialized for codes with cyclic parity-check matrices. We compare the results obtained with cyclic matrices to those obtained by non-cyclic representations. As a result, we find out that matrices of cyclic form are a preferable choice for parity-check matrices of cyclic codes.

We tacitly assume that only codewords of minimum weight from the dual code are used as parity checks [SV04].

C.1 Lower Bounds on the Stopping Redundancy Hierarchy of Cyclic Codes

As a first step, we specialize the lower bound derived in Proposition 6.2.15 to the case of parity-check matrices of cyclic form.

Proposition C.1.1 *Let κ be the offset between two cyclic shifts. The number of stopping sets of size σ resolved by a cyclic (redundant) parity-check matrix with dimensions $m \times n$, $n-k \leq m \leq n$, is given by*

$$|\bigcup_{i=1}^{m} \Sigma_\sigma\left(\boldsymbol{H}(i,:)\right)| \leq m\,|\Sigma_\sigma\left(\boldsymbol{H}(1,:)\right)| - \frac{2}{m} \cdot \sum_{\kappa=1}^{m-1} (m-\kappa) |\Sigma_\sigma\left(\boldsymbol{H}(1,:)\right) \cap \Sigma_\sigma\left(\boldsymbol{H}((1+\kappa) \bmod^* m,:)\right)|. \tag{C.1}$$

The operator $\bmod^*$ is introduced in Chapter 6 on page 77. Note that we assume in this proposition that the row indexed by $((\kappa+1) \bmod^* m)$ is the κ-th cyclic shift of the first row.

The result of Proposition C.1.1 is a special case of Equation (6.12). Observe two properties of cyclic parity-check matrices. First, each row resolves the same number of stopping sets. This

is deducted from the fact that the positions of the resolved stopping sets are cyclic shifts of each other. Second, the number of stopping sets simultaneously resolved by two rows only depends on their cyclic offset κ. Let the set $\mathcal{QQ}'_\kappa(\boldsymbol{H})$ denote all pairs of positions $(a, a+\kappa) \bmod^* n$ in a given row of $\boldsymbol{H}$, $1 \leq a \leq n$, with entry Q at position $a \bmod^* n$ and entry Q' at position $a+\kappa \bmod^* n$, $Q, Q' \in \{Z, O\}$. For parity-check matrices of cyclic form, the cardinalities of these sets equal the intersection numbers given in Definition 6.2.16, i.e. $|\mathcal{QQ}'_\kappa(\boldsymbol{H})| = |\mathcal{Q}_{a \bmod^* n} \cap \mathcal{Q}'_{a+\kappa \bmod^* n}|$, $\forall a \in \{1, \ldots, n\}$.

In the following, we present a specialization of Lemma 6.2.17.

Lemma C.1.2 *We calculate the number of stopping sets resolved jointly by two rows of a cyclic parity-check matrix with cyclic offset κ as*

$$|\Sigma_\sigma(\boldsymbol{H}(1,:)) \cap \Sigma_\sigma(\boldsymbol{H}((\kappa+1) \bmod^* m, :))| = |\mathcal{OO}_\kappa| \cdot \binom{|\mathcal{ZZ}_\kappa|}{\sigma-1} + |\mathcal{OZ}_\kappa| \cdot |\mathcal{ZO}_\kappa| \cdot \binom{|\mathcal{ZZ}_\kappa|}{\sigma-2}. \quad \text{(C.2)}$$

The result of Equation (C.2) can be calculated by determining the involved variables by simple counting arguments, i.e. by finding

$$|\mathcal{OO}_\kappa| = \sum_{j=\kappa+1}^{n+\kappa} \left\lfloor \frac{1}{2} \left(H(\ell, j \bmod^* n) + H(\ell, j-\kappa \bmod^* n) \right) \right\rfloor, \forall \ell,$$

$$|\mathcal{ZZ}_\kappa| = \sum_{j=\kappa+1}^{n+\kappa} \left\lfloor \frac{1}{2} \left((1 - H(\ell, j \bmod^* n)) + (1 - H(\ell, j-\kappa \bmod^* n)) \right) \right\rfloor, \forall \ell,$$

$$|\mathcal{ZO}_\kappa| = \sum_{j=\kappa+1}^{n+\kappa} \left\lfloor \frac{1}{2} \left(H(\ell, j \bmod^* n) + (1 - H(\ell, j-\kappa \bmod^* n)) \right) \right\rfloor, \forall \ell.$$

The variable $|\mathcal{OZ}_\kappa|$ is found in an analog way. Using Lemma 6.2.10 we obtain the number of stopping sets resolved by the first (or any other) parity check as

$$|\Sigma_\sigma(\boldsymbol{H}(1,:))| = \binom{n - \sum_{j=1}^{n} H(\ell, j)}{\sigma - 1} \cdot \sum_{j=1}^{n} H(\ell, j) \, \forall \ell.$$

Note that in all equations above it is assumed that $\binom{i}{j} = 0$ for $i < j$.

The expressions in Lemma C.1.2 are derived by simple counting arguments. We observe that the specialization on cyclic parity-check matrices allows for straightforward calculation of the parameters required for the bound in Equation (C.1). Be informed that due to the restricted choice of parity-check matrices, the derived lower bound can be underrun by realizations using matrices of non-cyclic form. One example for a lower bound is using the idempotent [MS77] of

the dual code and its cyclic shifts to create a parity-check matrix. The resulting lower bounds $\underline{\mathrm{B}}(\cdot)$ on the stopping redundancy hierarchy of the $[23, 12, 7]$ Golay code were derived in [HMLH08] and are given in the following.

	$\underline{\mathrm{B}}(\rho_4)$	$\underline{\mathrm{B}}(\rho_5)$	$\underline{\mathrm{B}}(\rho_6)$	$\underline{\mathrm{B}}(\rho_7)$
$[23, 12, 7]$ Golay	4	6	10	19

Table C.1: Lower bounds on the stopping redundancy hierarchy of the $[23, 12, 7]$ Golay code.

C.2 Upper Bounds on the Stopping Redundancy Hierarchy of Cyclic Codes

We derive upper bounds on the stopping redundancy hierarchy for the Golay $[23, 12, 7]$ code as well as various Hamming and BCH codes. These constructive bounds are used for two kinds of comparisons. First, we relate them to the theoretical upper bounds on the stopping redundancy hierarchy from Section 6.2.2. Second, we apply construction methods known from literature, which all result in check matrices of non-cyclic form, and compare these methods to our approach. To this end, we construct parity-check matrices avoiding stopping sets of given sizes with both methods and compare the number of required rows. Two main observations can be reported from this investigation. For non-redundant matrices, the decoding performance of matrices of cyclic and non-cyclic form is equal. To find this, we deployed the results of [AGW07], where the number of stopping sets of a full-rank, non-redundant parity-check matrix of a Hamming code was derived in analytical form. However, if the matrix has more than $n - k$ rows, the performance depends on the matrix. We show that the cyclic form is a preferable choice in this case by comparing it to the promising approach in [HT07]. Subsequently, we extend the approach in [HT07] to BCH codes and again compare to matrices of cyclic form.

C.2.1 The Binary Golay Code

We compare the performance of cyclic parity-check matrices of the $[23, 12, 7]$ Golay code $\mathcal{G}$ to general bounds from Section 6.2.2. As the choice of a cyclic parity-check matrix is not unique, we group the 506 codewords of minimum weight eight in the dual code $\mathcal{G}^\perp$ ($[23, 11, 8]$) into cyclic orbits and select one COG, cf. Section 5.1.1, to represent this orbit. As all parity-check equations in a cyclic orbit are cyclic shifts of each other, it is sufficient to estimate the performance of a matrix of cyclic form generated by the COG. For this code, we obtain 22 COGs representing 23 parity checks each[1].

[1]In Chapter 7 we extend the idea of grouping parity checks to *cog families*. This is however only relevant when $m = n$ shifts are used.

Different choices for the applied COG result in different upper bounds on the stopping redundancy hierarchy. We concentrate on three COGs that offer particularly good and poor performance, and refer to them by the subscripts A to C. The COG indexed by A has the octal representation $\text{cog}_{\text{Golay},[23,12,7],\text{A}} = [2\,1\,2\,1\,3\,5\,0\,0]$, where the MSB is located on the left-hand side. Its polynomial representation is $1+x^4+x^6+x^{10}+x^{12}+x^{13}+x^{14}+x^{16}$. The COGs indexed by B and C have the representations $\text{cog}_{\text{Golay},[23,12,7],\text{B}} = [2\,2\,1\,1\,1\,4\,6\,0]$ and $\text{cog}_{\text{Golay},[23,12,7],\text{C}} = [3\,4\,6\,0\,3\,2\,0\,0]$, respectively.

Table C.2 lists upper bounds $\overline{\text{B}}(\rho_j(\mathcal{C}))$ on the stopping redundancy hierarchy obtained from COGs $\text{cog}_{\text{Golay},[23,12,7],\text{A}}$, $\text{cog}_{\text{Golay},[23,12,7],\text{B}}$, and $\text{cog}_{\text{Golay},[23,12,7],\text{C}}$ of the parity-check matrix of the $[23, 12, 7]$ Golay code and the general upper bounds obtained from Theorem 6.2.21 with $\epsilon = 10^{-3}$ and from Equation (6.23).

	$\overline{\text{B}}(\rho_4(\mathcal{C}))$	$\overline{\text{B}}(\rho_5(\mathcal{C}))$	$\overline{\text{B}}(\rho_6(\mathcal{C}))$	$\overline{\text{B}}(\rho_7(\mathcal{C}))$
$\text{COG}_{\text{Golay},[23,12,7],\text{A}}$	$n-k$	16	18	23
$\text{COG}_{\text{Golay},[23,12,7],\text{B}}$	13	15	19	23
$\text{COG}_{\text{Golay},[23,12,7],\text{C}}$	$n-k$	16	21	> 23
Minimum	$n-k$	15	18	23
Upper bound acc. to Theorem 6.2.21, $\epsilon = 10^{-3}$	39	–	–	–
Upper bound acc. to Equation (6.23)	1023	1023	1023	1023

Table C.2: Constructive and theoretical upper bounds on the stopping redundancy hierarchy of the $[23, 12, 7]$ Golay code.

As can be seen from Table C.2, the stopping redundancy hierarchy of the Golay code is bounded by

$$\rho_4(\mathcal{C}) \leq n-k,\ \rho_5(\mathcal{C}) \leq 15, \rho_6(\mathcal{C}) \leq 18, \rho_7(\mathcal{C}) \leq 23.$$

The insight provided at this point is the fact that the theoretical bound in Equation (6.23) is extremely loose compared to the constructive bounds which are found by means of cyclic parity-check matrices. The applicability of the upper bound which is based on the high-probability variation of LLL, cf. Theorem 6.2.21, is limited to $\ell \leq \left\lfloor \frac{d+1}{2} \right\rfloor$. We observe that this bound is tighter when compared to Equation (6.23), but it is also outperformed by the constructive bounds.

C.2.2 Hamming Codes

Recently, Weber and Abdel-Ghaffar [WAG05] as well as Hollmann and Tolhuizen [HT07] introduced approaches for generating parity-check matrices of Hamming codes that resolve all erasure patterns which do not correspond to ML-dead-end sets up to a given size. The authors in [WAG05] contributed pioneering work when they introduced the notion of parity-check matrices, which allow to resolve all erasure patterns of weight three which do not correspond to codewords of this weight. Hollmann and Tolhuizen were inspired by these results and overcame

the weight confinement. As a result, they introduced the *generic* $(\bar{m}, \bar{\sigma})$ *erasure resolving sets*[2] $\mathcal{W}_{\bar{m},\bar{\sigma}}$. These sets are defined as

$$\{\boldsymbol{w} = [w_1, w_2, \ldots, w_{\bar{m}}] \mid w_1 = 1, \mathrm{wt}(\boldsymbol{w}) \leq \bar{\sigma}\}.$$

The collection of parity checks $\{\boldsymbol{w} \odot \boldsymbol{H} \mid \boldsymbol{w} \in \mathcal{W}_{\bar{m},\bar{\sigma}}\}$ resolves all erasure patterns up to a size of $\bar{\sigma}$ which do not correspond to ML-dead-end sets. In this context, $\bar{m}$ is chosen such that it equals the co-dimension of the code, what allows for the multiplication of $\boldsymbol{w}$ with a non-redundant parity-check matrix.

Let us first consider the class of $[2^b - 1, 2^b - b - 1, d = 3]$, $b \in \mathbb{N}$, Hamming codes. For these codes, $\bar{m} = \mathrm{ld}(n+1)$ holds. We use the approach in [HT07] to construct parity-check matrices that are generic $(\bar{m}, \bar{\sigma})$ erasure resolving sets, and then compare the stopping set distribution of these matrices with the distribution of cyclic parity-check matrices. For Hamming codes, $\rho_2(\mathcal{C}) = \rho_3(\mathcal{C}) = n - k$ holds, i.e. the stopping redundancies of interest are already obtained with parity-check matrices of standard size. Consequently, we compare the number of unresolved stopping sets of size $\sigma = 3$ for several fixed values of m.

In order to generate a non-cyclic parity-check matrix of a Hamming code, we follow [MS77] and define a row vector over the finite field $\mathbb{F}_{n+1}$, $n = 2^{n'} - 1, n' > 0$, namely

$$\boldsymbol{H}_{\mathrm{h,\,s}} = \begin{bmatrix} \varpi^{0 \cdot i} & \varpi^{1 \cdot i} & \ldots & \varpi^{(n-1) \cdot i} \end{bmatrix},$$

where ϖ denotes a primitive element of the underlying field. It is common practice to set $i = 1$. The binary parity-check matrix for the Hamming code is generated by representing the elements of $\boldsymbol{H}_{\mathrm{h}}$ as binary vectors over the vector space $\mathbb{F}_2^{\mathrm{ld}(n+1)}$, i.e. each element ϖ^f, $f \in \{0, \ldots, n-1\}$, is described by a binary column vector. This non-cyclic matrix has no redundant rows and is referred to as the *standard parity-check matrix* for Hamming codes. We will henceforth denote it by $\boldsymbol{H}_{\mathrm{h,\,s}}$.

Let us now construct a redundant matrix which consists of linear combinations of the rows of $\boldsymbol{H}_{\mathrm{h,\,s}}$. Start by constructing a generic erasure resolving set $\mathcal{W}_{\bar{m},\bar{\sigma}}$ with $\bar{m} = \mathrm{ld}(n+1)$ and $\bar{\sigma} = 3$ in order to consider all erasure patterns which do not correspond to ML-dead-end sets of size up to 3. This set is applied to the matrix as stated in [HT07], what results in a redundant parity-check matrix with

$$\sum_{i=0}^{\bar{\sigma}-1} \binom{\bar{m}-1}{i} \Bigg|_{\bar{m}=\mathrm{ld}(n+1),\,\bar{\sigma}=3} = \mathrm{ld}(n+1) + \binom{\mathrm{ld}(n+1)-1}{2} \tag{C.3}$$

rows. This matrix is denoted as $\boldsymbol{H}_{\mathrm{h,\,g}}$. All unresolved stopping sets of size three in such a parity-check matrix are codewords of the Hamming code. As a next step, we construct redundant parity-check matrices of cyclic form, obtained from m consecutive cyclic shifts of a minimum

[2]The original terminology used by the authors reads "generic erasure correcting sets". We use the term resolve, as erasure contain no wrong information and hence do not need to be corrected.

weight codeword of the dual code. In Table C.3 and Table C.4 we compare the numbers of unresolved stopping sets of size three in the redundant parity-check matrices described above, for $n = 63$ and $n = 127$. In this context, the COG vectors of the two cyclic parity-check matrices are

$$\mathrm{COG}_{\mathrm{Hamming},[63,57,3]} = [4\,1\,4\,2\,4\,7\,5\,0\,7\,1\,1\,3\,3\,5\,4\,6\,5\,3\,7\,4\,0],$$

and

$$\mathrm{COG}_{\mathrm{Hamming},[127,120,3]} = [1\,0\,4\,6\,1\,3\,5\,3\,3\,0\,1\,4\,6\,5\,1\,6\,3\,6\,6\,4\,1\,2\,5\,7\,5\,1\,2\,1\,5\,6\,1\,7\,7\,0\,3\,5\,7\,1\,3\,1\,1\,0\,0],$$

respectively. For the comparison, we choose the values $m = n - k$, $m = \sum_{i=0}^{\bar{\sigma}-1} \binom{\bar{m}-1}{i}$ (cf. Equation (C.3)), as well as several higher values.

	Number of unresolved stopping sets $\left\lvert\overline{\Sigma}_{\sigma}(\boldsymbol{H})\right\rvert$, $\sigma = 3$	
m	Parity-check matrix from [HT07]	Cyclic parity-check matrix
$6 = n - k$	2261	2261
16	651	655
17		653
18		651

Table C.3: Number of unresolved stopping sets of size $\sigma = 3$ ($[63, 57, 3]$ Hamming code).

	Number of unresolved stopping sets $\left\lvert\overline{\Sigma}_{\sigma}(\boldsymbol{H})\right\rvert$, $\sigma = 3$	
m	Parity-check matrix in [HT07]	Cyclic parity-check matrix
$7 = n - k$	11970	11970
22	2667	2672
26		2667

Table C.4: Number of unresolved stopping sets of size $\sigma = 3$ ($[127, 120, 3]$ Hamming code).

The results in Table C.3 and Table C.4 are unanimous. The redundant parity-check matrices of cyclic form obtain almost the decoding performance of the matrices constructed with the method in [HT07]. Be reminded that the latter approach is a method which was tailored for Hamming codes whereas the use of cyclic parity-check matrices is a general proceeding. For example, there are 11970 stopping sets of size $\sigma = 3$ in a full-rank parity-check matrix of the [127,120,3] Hamming code. This number can be reduced to 2667 stopping sets only, if one uses 22 rows of the construction described in [HT07], or 26 rows, when using cyclic parity-check matrices. This motivates a further study of cyclic codes in connection with parity-check matrices of cyclic form.

C.2.3 Double- and Triple-Error-Correcting BCH Codes

In this section, we extend our focus to the stopping redundancy hierarchy of BCH codes, in particular to the $[127, 113, 5]$ double-error-correcting BCH code, and the $[31, 16, 7]$ triple-error-correcting BCH code. We relate the stopping redundancy hierarchy of cyclic matrices to non-cyclic matrices and to general bounds. As a first step, we introduce a method to create well-performing non-cyclic parity-check matrices for double-error-correcting BCH codes. As our method extends the approach by Hollmann and Tolhuizen [HT07], we refer to it as the *generalized HT* (Hollmann-Tolhuizen) method.

The Generalized HT Method

This method follows basic schemes for constructing parity-check matrices by means of a primitive element of a finite field [MS77, p. 204]. In a nutshell, this algorithm uses the fact that the Hamming code is a super-code of the corresponding BCH code and constructs the matrix in four steps. First, a non-cyclic parity-check matrix for the Hamming code is created and redundancy is added. In order to accomplish this, we follow the method in [HT07]. Subsequently, additional parity checks are appended such that the set of valid codewords is restricted to codewords of the BCH code. A second and a third set of redundant rows are added in order to resolve all stopping sets corresponding to Hamming codewords and to reach a stopping distance of 5, respectively. In the following, we provide a detailed step-by-step description of the generalized HT method, consisting of 4 steps.

Step 1: This step creates a redundant parity-check matrix for a Hamming code using the method proposed by Hollmann and Tolhuizen, which allows for resolving all erasure patterns that do not correspond to ML-dead-end sets of size up to and including $\sigma = 3$. Obviously, all remaining stopping sets of that size correspond to codewords of the Hamming code. Furthermore, we can use this method to resolve erasure patterns of size 4 which do not include codewords of the Hamming code. Hence, we construct a generic erasure resolving set with $\bar{m} = \mathrm{ld}(n+1)$ and $\bar{\sigma} = 4$. The redundant matrix $\boldsymbol{H}_{\mathrm{h,\,g}}$ is created by multiplication of the standard parity-check matrix for the Hamming code, $\boldsymbol{H}_{\mathrm{h,\,s}}$ and the generic erasure resolving set $\mathcal{W}_{\bar{m},\bar{\sigma}}$ with $\bar{\sigma} = 4$. Figure C.1 visualizes the first step of the construction procedure.

Step 2: The parity-check matrix created in Step 1 is extended such that it forms a parity-check matrix of the BCH code which is a subset of the considered Hamming code and has minimum distance $d = 5$. This is done by appending the binary column-expansion of the row vector

$$\boldsymbol{h}_{\mathrm{BCH}} = \left[\begin{array}{cccc} \varpi^{0\cdot(1+i)} & \varpi^{1\cdot(1+i)} & \ldots & \varpi^{(n-1)(1+i)} \end{array} \right], \tag{C.4}$$

where $i = 2$ is commonly chosen [MS77, p.204] in order to construct BCH codes with minimum distance $d = 5$. We refer to this second component of the redundant parity-check matrix of the BCH code as $\boldsymbol{H}_{\mathrm{BCH}}$. Figure C.2 shows the second step of the construction procedure.

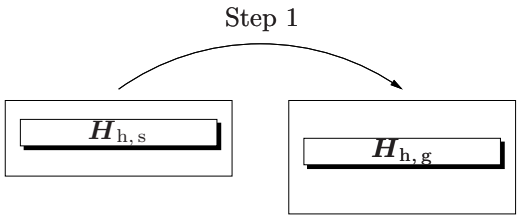


Figure C.1: Step 1: Transform $\boldsymbol{H}_{\mathrm{h,s}}$ into a generic $(\bar{m}, \bar{\sigma})$ erasure resolving set $\boldsymbol{H}_{\mathrm{h,g}}$.

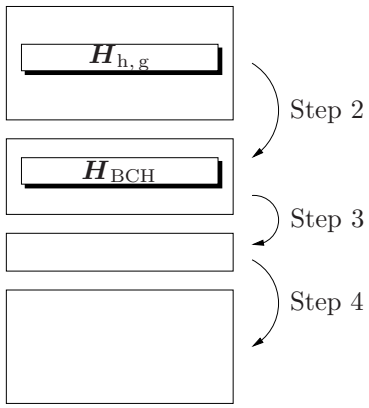


Figure C.2: Steps 2-4: Append $\boldsymbol{H}_{\mathrm{BCH}}$ (Step 2), and append further redundant checks (Steps 3 and 4).

Step 3: We add a second set of redundant rows such that all stopping sets of size $\sigma = 3$ and a subset of stopping sets of weight $\sigma = 4$ is resolved. We make use of the fact that all remaining stopping sets of that type correspond to or contain codewords of the Hamming code. In the following, we detail how this information is used to choose the redundant lines for a targeted removal.

Let $\boldsymbol{v}$ denote the support of an unresolved stopping set of size $\sigma = 3$ in vector form. On the one hand, the restriction of $\boldsymbol{H}_{\mathrm{h,g}}$ to $\boldsymbol{v}$ equals

$$\left[\left(\varpi^{a}\right)\ \left(\varpi^{b}\right)\ \left(\varpi^{a} \oplus \varpi^{b}\right)\right]$$

with $0 \leq a, b < n-2$, $\varpi^{a} \neq \varpi^{b}$, and on the other hand, the restriction of $\boldsymbol{H}_{\mathrm{BCH}}$ to $\boldsymbol{v}$ equals

$$\left[\left(\varpi^{3a}\right)\ \left(\varpi^{3b}\right)\ \left((\varpi^{a} \oplus \varpi^{b})^{3}\right)\right].$$

We partition the set of stopping sets of interest into two subclasses. Consider first all stopping sets for which $\varpi^{3a} = \varpi^{3b}$ holds, but $\varpi^{a} \neq \varpi^{b}$.

Lemma C.2.1 *([HMLH08]) In a finite field with characteristic two and order $n+1$, for any primitive element ϖ, $\varpi^{3a} = \varpi^{3b}$ implies $(\varpi^{a} \oplus \varpi^{b})^{3} = \varpi^{3a}$ iff $\varpi^{a} \neq \varpi^{b}$.*

Proof C.2.2 *([HMLH08]) As the field has characteristic two, it holds*

$$\begin{aligned}(\varpi^a \oplus \varpi^b)^3 &= \varpi^{3a} \oplus 3\varpi^{2a+b} \oplus 3\varpi^{a+2b} \oplus \varpi^{3b} \\ &= \varpi^{2a+b} \oplus \varpi^{a+2b}.\end{aligned}$$

The notation 3ϖ is symbolic for $\varpi \oplus \varpi \oplus \varpi$. Consequently,

$$(\varpi^a \oplus \varpi^b)^3 = \varpi^{3a} \odot \left(\varpi^{b-a} \oplus \varpi^{a-b}\right). \tag{C.5}$$

We still need to show $\left(\varpi^{b-a} \oplus \varpi^{a-b}\right) = 1$ in order to complete the proof. As $3a - 3b = j\,n$, for some integer $j \geq 0$, we conclude $a - b = j\,n/3$ and $b - a = 2j\,n/3$. In order for $\varpi^a \neq \varpi^b$ to hold, j has to be restricted to $j \neq 0 \,(\mathrm{mod}\, 3)$. For these values of j, one has $\left(\varpi^{b-a} \oplus \varpi^{a-b}\right) = \left(\varpi^{n/3} \oplus \varpi^{2n/3}\right)$. We proof the following more general claim. For each $\kappa \in \{2k+1 \mid k \in \mathbb{N}\}$, it holds that $\bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{in}{\kappa}} = 1$. This result includes the result we are interested in, $\kappa = 3$. Next we show that

$$\left(\bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{in}{\kappa}}\right)^2 = \bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{2in}{\kappa}} = \bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{in}{\kappa}}, \tag{C.6}$$

what is equivalent to $\bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{in}{\kappa}} = 1$. It is to observe that

$$\begin{aligned}\bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{2in}{\kappa}} &= \bigoplus_{i=1}^{\frac{(\kappa-1)}{2}} \varpi^{\frac{2in}{\kappa}} \oplus \bigoplus_{i=\frac{(\kappa+1)}{2}}^{\kappa-1} \varpi^{\frac{2in}{\kappa}} \\ &= \bigoplus_{i=1}^{\frac{(\kappa-1)}{2}} \varpi^{\frac{2in}{\kappa}} \oplus \bigoplus_{i'=1}^{\frac{(\kappa-1)}{2}} \varpi^{2(i'+\frac{\kappa}{2}-\frac{1}{2})\frac{n}{\kappa}} \\ &= \bigoplus_{i=1}^{\frac{(\kappa-1)}{2}} \varpi^{\frac{2in}{\kappa}} \oplus \bigoplus_{i'=1}^{\frac{(\kappa-1)}{2}} \varpi^{\frac{(2i'-1)n}{\kappa}} \\ &= \bigoplus_{i=1}^{\kappa-1} \varpi^{\frac{in}{\kappa}},\end{aligned}$$

where we have substituted $i' = i - \kappa/2 + 1/2$ and used the fact that $\varpi^n = 1$. ∎

Let us now use these results to generate appropriate redundant parity-check equations. Assume that $n/3$ is an integer. The number of stopping sets corresponding to Hamming codewords of weight three, for which $\varpi^{3a} = \varpi^{3b}$ holds, reads $n/3$, as only the parameter a can be chosen arbitrarily and in the range $0 \leq a < n/3$. The variable b depends on a and needs to fulfill $n/3 \leq b < n$. Next, we switch to the binary representation of $\boldsymbol{H}_{\mathrm{h,\,g}}$ and $\boldsymbol{H}_{\mathrm{BCH}}$ and consider their restrictions to $\boldsymbol{v}$. As all Hamming codewords need to fulfill the corresponding parity checks, the restriction of $\boldsymbol{H}_{\mathrm{h,\,g}}$ to $\boldsymbol{v}$ only contains rows of weight zero or two. As shown above, the same

restriction of $\boldsymbol{H}_{\mathrm{BCH}}$ only contains rows of weight zero or three. We state that each stopping set corresponding to a weight-three codeword of the Hamming code with $\varpi^{3a} = \varpi^{3b}$ can be resolved by a linear combination of one row from $\boldsymbol{H}_{\mathrm{h,\,g}}$ of weight two and one row of $\boldsymbol{H}_{\mathrm{BCH}}$ of weight three. The search for the appropriate combinations is performed by greedy computer search which aims at resolving as many stopping sets as possible with each linear combination. Figure C.2 visualizes this step.

Step 4: The final step resolves all stopping sets not considered by the preceeding steps. This includes stopping sets of weight 3 if $n/3$ is not an integer and stopping sets which do not correspond to Hamming codewords of weight three. In this case, we use greedy computer search techniques to identify appropriate redundant parity checks, which are appended to the parity checks found so far. This step is also shown in Figure C.2.

We later observe that this method compares favorably to the direct application of the method described in [HT07] to BCH codes and the general bounds discussed so far. However, cyclic codes significantly outperform this method.

Cyclic Parity-Check Matrices of Double- and Triple-Error-Correcting BCH Codes

We create cyclic parity-check matrices for the $[127, 113, 5]$ BCH code and the $[31, 16, 7]$ BCH code.

The dual of the $[127, 113, 5]$ BCH code has 4572 codewords of minimum weight 56, which can be separated into 36 cyclic orbits, each represented by a COG as listed below. In particular, we focus on four selected COGs, indexed by A through D, and analyze their underlying parity-check matrices in detail.

$$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{A}} = [1\,7\,6\,4\,0\,3\,0\,6\,5\,4\,4\,5\,4\,0\,7\,5\,0\,4\,5\,4\,7\,6\,5\,1\,6\,1\,6\,0\,2\,0\,4\,2\,6\,5\,2\,4\,2\,4\,4\,0\,0\,5\,6],$$

$$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{B}} = [1\,7\,2\,4\,2\,5\,0\,2\,6\,1\,2\,1\,5\,4\,1\,1\,1\,1\,5\,2\,6\,1\,0\,7\,2\,1\,2\,5\,5\,1\,6\,1\,4\,0\,4\,6\,5\,4\,1\,4\,2\,7\,4],$$

$$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{C}} = [1\,7\,5\,2\,6\,5\,5\,3\,3\,6\,4\,6\,1\,3\,1\,2\,6\,4\,2\,1\,0\,7\,1\,1\,7\,0\,4\,0\,2\,4\,0\,2\,5\,4\,0\,3\,0\,4\,5\,2\,2\,4\,2],$$

$$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{D}} = [1\,7\,5\,1\,7\,0\,3\,1\,2\,5\,2\,6\,7\,3\,4\,6\,5\,0\,2\,1\,0\,2\,0\,7\,0\,3\,6\,5\,4\,0\,6\,1\,2\,2\,1\,0\,1\,4\,3\,0\,6\,4\,4].$$

We partition the dual of the $[31, 16, 7]$ BCH code with 465 codewords of minimum weight eight into 15 cyclic orbits. Four representative COGs, indexed by A through D, are listed below.

$$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{A}} = [1\,4\,1\,4\,0\,5\,0\,0\,0\,2\,2],$$

$$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{B}} = [1\,4\,0\,6\,1\,0\,4\,1\,0\,2\,0],$$

$$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{C}} = [1\,5\,0\,0\,0\,5\,0\,0\,4\,1\,4],$$

$$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{D}} = [1\,5\,0\,4\,0\,2\,0\,0\,1\,3\,0].$$

In a next step, we compare redundant matrices of cyclic form to the generalized HT method and the straightforward application of the approach in [HT07] as well as the theoretical bound in Equation (6.23) from Section 6.2.2.

	$\overline{\mathrm{B}}(\rho_4(\mathcal{C}))$	$\overline{\mathrm{B}}(\rho_5(\mathcal{C}))$
$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{A}}$	20	34
$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{B}}$	22	34
$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{C}}$	23	46
$\mathrm{COG}_{\mathrm{BCH},[127,113,5],\mathrm{D}}$	22	45
Minimum	20	34
Eq. (6.23)	469	469

Table C.5: Upper bounds on the stopping redundancy hierarchy of the $[127, 113, 5]$ BCH code.

	$\overline{\mathrm{B}}(\rho_4(\mathcal{C}))$			
Code	Cyclic matrix	Generalized HT	HT, $\bar{m} = 14$, $\bar{\sigma} = 3$	General bound [SV06]
$[127, 113, 5]$	20	96	92	469
	$\overline{\mathrm{B}}(\rho_5(\mathcal{C}))$			
Code	Cyclic matrix	Generalized HT	HT, $\bar{m} = 14$, $\bar{\sigma} = 4$	General bound [SV06]
$[127, 113, 5]$	34	229	378	469

Table C.6: Upper bounds on $\rho_\ell(\mathcal{C})$, $\ell \leq 5$.

The values presented for cyclic codes are obtained by searching for the best COG consisting of minimum weight codewords. The COG indexed by A has the "best" stopping set properties, when both the numbers of stopping sets of size three and four are considered. Upper bounds on the stopping redundancy hierarchy for all four COGs are given in Table C.5. As a consequence, for the $[127, 113, 5]$ BCH code one has

$$\rho_4(\mathcal{C}) \leq 20,\ \rho_5(\mathcal{C}) \leq 34.$$

Table C.6 again confirms that the general bounds derived in [SV06] are not tight. Furthermore it shows that the generalized HT method can outperform the direct application of the method from [HT07] to BCH codes. The sub-optimality of this approach for $\bar{\sigma} \geq 3$ and large enough

$\bar{m}$ is already stated in [HT07]. The most prominent observation is the significant superiority of the cyclic (redundant) parity-check matrix construction to the generalized HT method and the direct application of the method in [HT07].

The results of a similar analysis performed on the $[31, 16, 7]$ BCH code are shown in Table C.7. As can be seen from the table, the stopping redundancy hierarchy of the $[31, 16, 7]$ BCH codes

	$\overline{\mathrm{B}}(\rho_4(\mathcal{C}))$	$\overline{\mathrm{B}}(\rho_5(\mathcal{C}))$	$\overline{\mathrm{B}}(\rho_6(\mathcal{C}))$	$\overline{\mathrm{B}}(\rho_7(\mathcal{C}))$
$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{A}}$	$n-k=15$	18	19	21
$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{B}}$	$n-k=15$	16	20	22
$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{C}}$	$n-k=15$	$n-k=15$	20	28
$\mathrm{COG}_{\mathrm{BCH},[31,16,7],\mathrm{D}}$	$n-k=15$	16	21	26
Minimum	$n-k=15$	$n-k=15$	19	21
Upper bound (Theorem 6.2.21, $\epsilon = 10^{-3}$)	45	–	–	–
Upper bound (Eq. (6.25), Eq. (6.23))	91	175	4943	4943

Table C.7: Upper bounds on the stopping redundancy hierarchy of the $[31, 16, 7]$ BCH code.

is upper bounded by

$$\rho_4(\mathcal{C}) = 15,\ \rho_5(\mathcal{C}) = 15,\ \rho_6(\mathcal{C}) \leq 19,\ \rho_7(\mathcal{C}) \leq 21.$$

Appendix D

Decoding Performance and Structural Delay of Algebraic Codes

This section provides selected results for algebraic codes of rates equal or close to one half, where the standard BP decoder and the (L)-MBBP[1] setup are used. It is well-known that algebraic codes are not designed for BP decoding. As a consequence, comparably high SNRs are required to obtain given target error rates. Significant performance improvements can be obtained by using the (L)-MBBP approach. For the simulation results shown in this section, a maximum number of $I_{\max} = 100$ iterations is set for all algebraic codes. The codes are the $[24, 12, 8]$ extended Golay code, detailed in Section 7.5 and [MS77, Ch. 20], the $[31, 16, 7]$ BCH code as discussed in Section 7.3 and [MS77, Ch. 9], the $[48, 24, 12]$ extended QR code [MS77, Ch. 16], and the $[63, 39, 9]$ BCH code, cf. Section 7.6.2. The MBBP-NX-S decoding approach, cf. Section 7.2, is used in the multiple-bases decoding approach. If additional improvements can be gained, the probability based variation of the Leaking approach is added, where $p_{\mathcal{L}} = 0.98$ and $I'_{\max} = 300$ is set. It is assumed in all cases that SPNSE (see Section 8.1.1) is used.

Table D.1 lists the required SNRs for the error rates FER $= 10^{-3}$ and BER $= 10^{-5}$. Figure D.1 shows the corresponding results. For comparison, we show the results for PEG-optimized LDPC codes with BP and L-MBBP decoding, respectively, where at most $I'_{\max} = 200$ iterations were allowed. Also, the results obtained by convolutional codes as well as information theoretic bounds are presented. For the considered algebraic codes, a high gain of up to about 0.9 dB is obtained

		BER $= 10^{-5}$		FER $= 10^{-3}$	
Code	l_{MBBP}	BP	(L)-MBBP	BP	(L)-MBBP
$[24, 12, 8]$ ext.-Golay code	11	6.71 dB	5.67 dB	5.33 dB	4.60 dB
$[31, 16, 7]$ BCH code	15	6.09 dB	5.52 dB	5.01 dB	4.39 dB
$[48, 24, 12]$ ext.-QR code	23	6.05 dB	5.15 dB	5.46 dB	4.53 dB
$[63, 39, 9]$ BCH code	$2 \cdot 6$	5.51 dB	4.73 dB	4.79 dB	4.09 dB

Table D.1: Required SNR ($10 \cdot \log_{10}(E_{\text{b}}/N_0)$) to obtain a given error rate for algebraic codes.

[1]The parentheses indicate that Leaking is only used in some but not all cases.

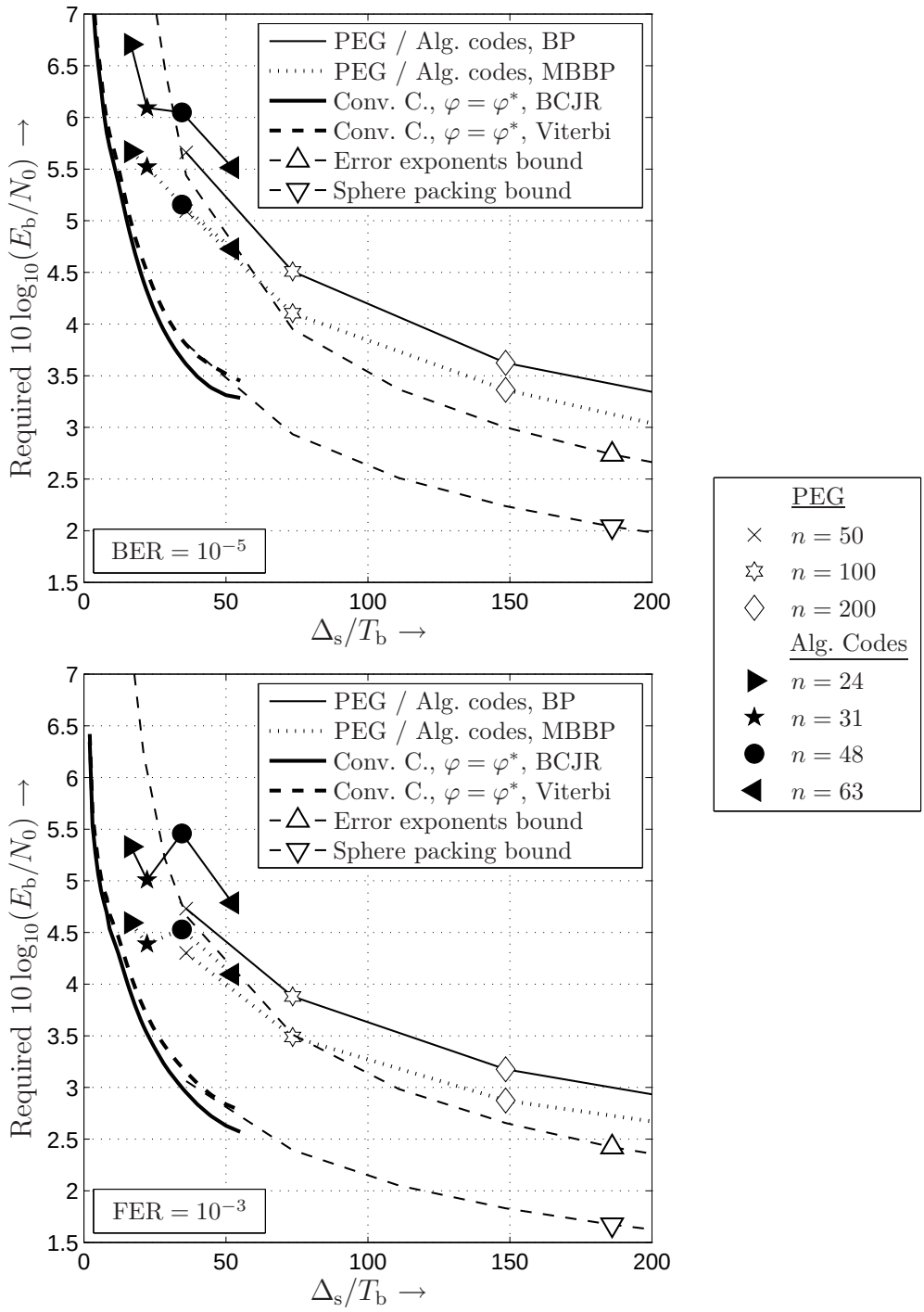


Figure D.1: Required SNR to obtain BER $= 10^{-5}$ and FER $= 10^{-3}$, respectively. Results for BP decoding and MBBP decoding are given. We set $I_\mathrm{max} = 100$ and $p_\mathcal{L} = 0.98$ for the algebraic codes. For decoding the PEG-optimized codes, $I_\mathrm{max} = 200$ and $p_\mathcal{L} = 0.9$ are set.

when (L)-MBBP is used. This can be attributed to the dense parity-check matrix representations of the algebraic codes which prevent good decoding performance on the standard BP algorithm. It is a remarkable result that algebraic codes using (L)-MBBP outperform PEG-optimized codes of comparable structural delay, if the latter class of codes is decoded by standard BP. Further, the algebraic codes almost obtain the performance of PEG codes in connection with (L)-MBBP decoding. This is not in general extendable to algebraic codes of longer length.

Appendix E

List of Acronyms and Variables

List of Acronyms

The acronyms used throughout this thesis together with their meaning and the page of first occurrence are listed in the following table.

Acronym	*Meaning*	*Page*
APP	A-posteriori probability	23
ASK	Amplitude-Shift Keying	5
AWGN	Additive White Gaussian Noise	3
BCH	Bose-Chaudhuri-Hocquenghem	38
BEC	Binary Erasure Channel	3
BER	Bit Error Rate	46
BISMC	Binary Input Symmetric Memoryless Channel	13
BlP	Block Processing	125
BMD	Bounded Minimum Distance	73
BP	Belief Propagation	21
BPSK	Binary Phase-Shift Keying	9
BSC	Binary Symmetric Channel	72
CDS	Communicating-Decoders Selector	102
COG	Cyclic Orbit Generator	38
ECB	Equivalent Complex Baseband	5
ECC	Error Control Coding	9
ETSI	European Telecommunications Standards Institute	45
FEC	Forward Error Correction	9
FER	Frame Error Rate	46
GCD	Greatest Common Divisor	97
IEEE	Institute of Electrical and Electronics Engineers	2
IID	Independent, Identically Distributed	4
LDPC	Low-Density Parity-Check	1
LIS	Least-Iteration Selector	101
LLL	Lovász Local Lemma	68
LLR	Log-likelihood Ratio	16
L-MBBP	Leaking MBBP	111
LMS	Least-Metric Selector	93

MAP	Maximum A-Posteriori	36
MBBP	Multiple-Bases Belief-Propagation	92
ML	Maximum Likelihood	36
MLSE	Maximum-Likelihood Sequence Estimation	52
MLSSE	Maximum-Likelihood Symbol-by-Symbol Estimation	52
MSB	Most Significant Bit	51
PAM	Pulse Amplitude Modulation	5
PD	Permutation Decoding	75
PDF	Probability Density Function	6
PEG	Progressive Edge-Growth	41
PSK	Phase-Shift Keying	5
QAM	Quadrature-Amplitude Modulation	5
QPSK	Quadrature Phase-Shift Keying	7
QR	Quadratic-residue	94
RV	Random Variable	4
SAD	Stopping Automorphism Group Decoding	79
SG	Smart Guessing	79
SP	Serial Processing	123
SPB	Sphere Packing Bound	49
SPC	Single-Parity-Check	19
SPNSE	Serial Processing, Non-Systematic Encoding	123
SPSE	Serial Processing, Systematic Encoding	124
WiMAX	Worldwide Interoperability for Microwave Access	45

List of Variables

Random Variables

Variable	*Meaning*	*Page*
A	Mapped signal, RV, realization a	5
U	Source signal, RV, realization u	10
$\hat{U}$	Estimated source signal, RV, realization $\hat{u}$	10
X	Encoded signal, RV, realization x	4
$\hat{X}$	Estimated encoded signal, RV, realization $\hat{x}$	10
Y	Received signal, RV, realization y	10
Z	Noise signal, RV, realization z	6

Sets

Variable	*Meaning*	*Page*
$\mathcal{A}$	Signal constellation	5
$\mathcal{A}_i$	Decoders used for data communication (v_i, MBBP-X)	103
$\mathrm{Aut}(\mathcal{C})$	Automorphism group of $\mathcal{C}$	72
$\mathcal{B}$	Set of event sets	65
$\mathcal{C}$	Channel code, i.e. set of all codewords	9
$\mathcal{C}^{\perp}$	Dual of the channel code $\mathcal{C}$	35
$\underline{\mathcal{C}}$	Set of check nodes	21
$\mathcal{D}$	Set of cyclic orbit generators	96
$\mathcal{E}$	Event set	65
$\mathcal{F}$	Family of cyclic orbit generators	96
$\mathcal{I}$	Subset, general purpose	65
$\mathcal{L}_I$	Set of instructed variable nodes in iteration I (Leaking)	111
$\mathcal{O}$	Set of positions of ones	67
$\mathcal{P}$	Complete set of positions within a codeword	28
$\mathcal{P}'$	General-purpose subset of $\mathcal{P}$	46
$\mathcal{P}_\sigma$	Subset of $\mathcal{P}$ corresponding to a stopping set	59
$\mathcal{P}_{\mathrm{syst}}$	Set of positions of source symbols in a codeword (syst. enc.)	46
$\mathcal{P}_{\mathrm{par}}$	Set of parity positions in a codeword (syst. enc.)	73
$\mathcal{P}_{\mathrm{inv}}$	Set of positions (full-rank generator matrix)	46
$\mathcal{Q}$, $\mathcal{Q}'$	Placeholder for sets $\mathcal{O}$ and $\mathcal{Z}$	67
$\mathcal{S}$	Set of states of a convolutional code	50
$\Sigma_\sigma(\boldsymbol{H}(i,:))$	Set of stopping sets of size σ resolved by i-th row of $\boldsymbol{H}$	64
$\overline{\Sigma}_\sigma(\boldsymbol{H})$	Set of unresolved stopping sets of size σ when using $\boldsymbol{H}$	83
$\mathcal{T}_{c,j}$	j-th set of check-node indices connected by cycle of length c	116
$\mathcal{U}_i$	i-th set of PEG-code ensembles	43
$\mathcal{V}$	Set of indices of converged decoders in MBBP	93
$\underline{\mathcal{V}}$	Set of variable nodes	21
$\mathcal{W}_{\bar{m},\bar{\sigma}}$	Generic erasure resolving set	155
$\mathcal{X}$	Binary input alphabet	4

$\mathcal{Y}$	Set of subcodes	71
$\mathcal{Z}$	Set of positions of zeros	67
$\mathcal{ZZ}_i$, $\mathcal{OO}_i$, $\mathcal{ZO}_i$	Set of pairs of positions $a, a+i$	152

Numbers

Variable	*Meaning*	*Page*
$\mathbb{C}$	Complex numbers	5
$\mathbb{F}_q$	Galois field of order q	18
$\mathbb{R}$	Real numbers	8
$\mathbb{R}^+$	Positive-valued real numbers	72
$\mathbb{N}$	Natural numbers (not including zero)	38
$\mathbb{Z}$	Integers	4

Symbols

Variable	*Meaning*	*Page*
A_i	Number of codewords of weight i	48
c_j	Check node j	21
c	Length of a local cycle	116
C	Channel Capacity	4
d	Minimum distance of a linear block code	10
$\hat{d}$	Estimated minimum distance of a code	49
d_{free}	Free distance of a convolutional code	52
$d^{\perp}$	Dual distance of a linear block code	35
Δ_{p}	Processing delay	138
Δ_{s}	Structural delay	122
Δ_{t}	Total delay	138
E_{b}	Energy per bit	10
E_{s}	Energy per symbol	8
E	Event in a probability space	65
$\overline{E}$	Counter event to event E	69
ε	Erasure probability	4
η_i	Normalization factor in BP algorithm	34
f	Family indicator	96
f_{c}	Carrier frequency	5
F	Number of families	96
F_j	Factor node j	29
φ	Multiplication factor ($w_{\boldsymbol{\beta}}$, w_{p})	126
$g_{i,\text{oct}}$	Specification of convolutional code, octal	51
g_i	Specification of convolutional code, binary	51
$\mathcal{G}$	Graph	21
h	Cardinality of cyclotomic coset with sp. properties	77
I	Iteration number	27
I_ℓ	Number of iterations requ. by the ℓ-th decoder (MBBP)	100

$I_{\max}$	Maximum number of iterations	27
$I_{\text{sync,MBBP}}$	Iteration specifying the decoder synchronization	101
k	Code dimension of linear block code	10
$l_{\max}$	Maximum variable-node degree	39
l_{MBBP}	Number of representations in an MBBP setup	93
ℓ^*	Selected index, MBBP setup	101
L_i	Log-likelihood ratio, corresponding to p_i	16
L_{F}	Frame length for estimating the FER	55
m	Number of parity checks in a Tanner graph or matrix	21, 36
$\bar{m}$	Number of parity checks (generic erasure resolving set)	155
$\mu_{\text{F}_j \to \text{v}_i}$	Message from factor node F_j to variable node v_i	30
$\mu_{\text{v}_i \to \text{F}_j}$	Message from variable node v_i to factor node F_j	30
n	Code length of linear block code	10
$N_0/2$	Two-sided power spectral density of AWGN	6
ω	Constant codeword weight	67
ω_{r}	Weight of redundant codeword	116
p	Probability, generally used	14
p^{app}	A-posteriori probability (variable node)	23
p^{ext}	Extrinsic probability (variable node)	22
p^{int}	Intrinsic probability (variable node)	22
$p_{\mathcal{L}}$	Probability of an instructed variable node (Leaking)	113
ϖ	Primitive element	155
$\psi^{(\text{t})}$, $\psi^{(\text{r})}$, $\psi^{(\text{y})}$	Thresholds for Leaking Algorithm	113
$\psi_{\text{L}_i(x),\text{R}_i(x)}$	Threshold for norm. degree distribution pair $\text{L}_i(x), \text{R}_i(x)$	43
q^{app}	A-posteriori probability (check node)	23
q^{ext}	Extrinsic probability (check node)	22
Q, Q'	Placeholders for one and zero in intersection numbers	67
r	Result of parity-check equation	18
$r_{\max}$	Maximum check-node degree	39
R_{c}	Rate of the channel coding scheme	10
R	Rate of the transmission system	10
ρ_{D}	Parameter in Gallager bound	48
$\rho_d(\mathcal{C})$	Stopping redundancy	62
$[\rho_1(\mathcal{C}),\ldots,\rho_d(\mathcal{C})]$	Stopping redundancy hierarchy	63
S	Signal power	8
$S_s^\star(\mathcal{C})$	s-automorphism redundancy	81
$S^\star(\mathcal{C})$	Automorphism redundancy	81
σ	Size of a stopping set	59
$\bar{\sigma}$	Size of a stopping set (generic erasure resolving set)	155
σ_{z}^2	Noise power (AWGN channel)	6
σ_{R}^2	Noise power (AWGN channel), relevant for BPSK	7
t	Continuous time	5
t_d	Number of errors correctable by BMD, min. distance d	73
T_{b}	Duration for signaling one bit of information	9
T_{s}	Duration for signaling one transmission symbol (PAM)	6
τ_{b}	Index for time step of length T_{b}	122
$\lvert\mathcal{T}_c\rvert$	Number of sets of type $\mathcal{T}_{c,j}$	116
v_i	Variable node i	21

$\bar{v}_i$	Weighted majority vote (var. node v_i, MBBP-X)	103
w	Expansion factor for lifting (WiMAX codes)	45
$w_{\boldsymbol{\beta}}$	Number of time steps for backward state estimation, BCJR	53
w_{p}	Number of time steps (path estimation, Viterbi)	54

Auxiliary Symbols and Variables

Variable	*Meaning*	*Page*
a	Auxiliary variable at different positions	158
b	Auxiliary variable at different positions	65, 78
D	Denominator	15
δ_1	Number of parallel channels	13
δ_2	Number of symbols in mod 2 sum	18
ϵ	Parameter for Lovász Local Lemma	69
f_{R}	Auxiliary variable for systematic encoding	124
i, j, κ, ℓ	Counters and auxiliary variables at several places	
s	Running parameter, mostly used for stopping set sizes	75
θ	Half-angle specifying a cone (sphere packing bound)	49
θ_γ	Offset in mapping function γ	97
θ_σ	Auxiliary parameter for stopping redundancy	64
ξ	Auxiliary variable for code polynomials	78
Ξ	Group of permutations	83
W	Auxiliary variable in sphere packing bound	49

Time Processes

Variable	*Meaning*	*Page*
$s(t)$	Transmission signal, ECB domain	5
$s_{\mathrm{RF}}(t)$	Radio-frequent transmission signal	5
$y(t)$	Continuous-time received signal	6
$z(t)$	Continuous-time noise	6

Vectors and Matrices

Variable	*Meaning*	*Page*
$\boldsymbol{a}$	Vector of transmission symbols, length n	10
$\boldsymbol{\alpha}_i$	State probabilities, time step i (BCJR), length $\lvert\mathcal{S}\rvert$	53
$\boldsymbol{\beta}_i$	State probabilities, time step i (BCJR), length $\lvert\mathcal{S}\rvert$	53
$\boldsymbol{e}$	Vector of binary error symbols (BSC), length n	10
$\boldsymbol{G}$	Generator matrix, size $k \times n$	35
$\boldsymbol{h}_{\mathrm{BCH}}$	"BCH part" of a parity-check matrix (in vectorial form)	157
$\boldsymbol{H}$	Parity-check matrix, size $m \times n$ (standard: $m = n - k$)	21
$\boldsymbol{H}_{\mathrm{h,s}}$	Standard parity-check matrix of Hamming code	155

$\boldsymbol{H}_{\mathrm{h,g}}$	Generic erasure resolving set for Hamming code	155
$\boldsymbol{H}_{\mathrm{BCH}}$	"BCH part" of a parity-check matrix for the BCH code	157
$\boldsymbol{P}$	Parity part of $\boldsymbol{H} = [\boldsymbol{I}_{n-k} \mid \boldsymbol{P}]$	73
$\boldsymbol{s}$	Syndrome, length $n-k$	73
$\boldsymbol{u}$	Vector of source symbols, length k	10
$\boldsymbol{v}$	Stopping sets corr. to Hamming codewords, weight 3	158
$\boldsymbol{w}$	Vector of generic erasure resolving set	155
$\boldsymbol{x}$	Vector of encoded symbols, length n	10
$\boldsymbol{y}$	Vector of received symbols, length n	10
$\boldsymbol{M}(:, \mathcal{P}')$	Matrix access: restriction of $\boldsymbol{M}$ to $\mathcal{P}'$	46
$\boldsymbol{M}(\mathcal{P}', :)$	Matrix access: restriction of $\boldsymbol{M}^{\mathsf{T}}$ to $\mathcal{P}'$	60
$\boldsymbol{x}(\mathcal{P}')$	Vector access: subset of elements of $\boldsymbol{x}$, indexed by $\mathcal{P}'$	28

Functions

Variable	*Meaning*	*Page*
$\alpha(\cdot)$	Permutation, cyclic shift to the right	77
$\alpha_{\mathrm{W}}(\cdot)$	Permutation, suggested by Wolfmann	84
α_{bw}	Bandwidth extension factor	8
$\mathrm{A}(\cdot)$	Weight enumerator of a code	48
$\overline{\mathrm{B}}(\cdot)$	Upper bound	44
$\underline{\mathrm{B}}(\cdot)$	Lower bound	153
$\mathrm{bi2re}(\cdot)$	Transition from $\mathbb{F}_2$ to $\mathbb{R}$	27
$\beta(\cdot)$	Permutation, position i shifted to $2i \bmod n$	78
$\gamma(\cdot)$	Affine permutation	97
$\gamma_{\mathrm{i}}(\cdot)$	Affine permutation (index)	97
$\delta[\cdot]$	Discrete unit impulse	77
$\mathrm{E}_0(\cdot, \cdot, \cdot)$	Gallager function	48
$\mathrm{E}_{\mathrm{r}}(\cdot)$	Random coding exponent	48
$\mathrm{f}(\boldsymbol{x})$	Function, factorizable w.r.t. $\boldsymbol{x}$	28
$\mathrm{f}^{[\mathrm{M}]}(x_i)$	Marginalization of $\mathrm{f}(\boldsymbol{x})$ w.r.t. x_i	29
$\mathrm{f}^{(j)}(x)$	j-th derivative of $\mathrm{f}(x)$ for x	38
$\Phi_{zz}(\cdot)$	Power spectral density of $z(t)$	6
$\mathrm{g}(\cdot)$	Basic impulse (PAM)	6
$\mathrm{gi}(\cdot)$	Girth	26
$\mathrm{h}(\cdot)$	Monic parity-check polynomial of a cyclic code	37
$\mathrm{hyp}(\mathcal{X}, \mathcal{P}')$	Hypothetical set from alphabet $\mathcal{X}$ on set of positions $\mathcal{P}'$	33
$\mathrm{ind}(\cdot)$	Set of indices of items in set	65
$\mathrm{inv}(\cdot)$	Inverse of square matrix	46
$\mathrm{lgd}_{\mathcal{G}}(\cdot)$	Local girth distribution	40
$\mathrm{lgi}(\cdot, \cdot)$	Local girth of variable node in a graph	26
$\mathrm{L}(\cdot)$	Norm. degree distr. from variable node persp.	39
$\lambda(\cdot)$	Norm. degree distr. from edge persp., left-hand side	39
$\Lambda(\cdot)$	Degree distr. from variable node persp.	39
$\mathrm{map}(\cdot)$	Mapping function, general and restricted to BPSK	5, 9
$\mathcal{N}(\cdot)$	Neighborhood of a node (indices)	23
$\Pr(\cdot)$	Probability operator	4

$\mathrm{PDF}_X(x)$	Probability density function of cont. random variable X	6
$\mathrm{PD}_s(\mathrm{Aut}(\mathcal{C}))$	Subset of automorphism group (permutation dec.)	75
$\mathrm{R}(\cdot)$	Norm. degree distr. from the check node perspective	39
$\rho(\cdot)$	Norm. degree distr. from edge persp., right-hand side	39
$\mathrm{P}(\cdot)$	Degree distribution from check node perspective	39
$\mathrm{sub}_i(\cdot)$	Set of all subsets of input set, size i	65
$\mathrm{SAD}_s(\mathrm{Aut}(\mathcal{C}), \boldsymbol{H})$	Subset of automorphism group (automorphism dec.)	81
$\sum_{\sim(\cdot)}$	Summation over all variables except the specified one	29
$\mathrm{wt}(\cdot)$	(Hamming) weight of vector $\boldsymbol{x}$	63
$x_1 \bmod x_2$	Standard modulo function	38
$x_1 \bmod^* x_2$	Non-Standard modulo function, returns x_1 if $x_1 = x_2$	77
$\zeta(\cdot)$	Permutation	42

Operators

Operator	*Meaning*	*Page*
$\otimes$	Normalized multiplication	15
$\oslash$	Normalized division	25
$\oplus$	Summation in Galois field	18
$\odot$	Multiplication in Galois field	27
$(\cdot)^{\mathsf{T}}$	Transposed matrix or vector	27
$(\cdot)^*$	Conjugate complex	6
$(\cdot) \star (\cdot)$	Convolution	5
$\lvert\cdot\rvert$	Cardinality of a set and abs. value of a scalar number	5, 93
$\mathrm{E}\{\cdot\}$	Expectation operator, index specifies averaging ensemble	44
$\mathcal{H}\{\cdot\}$	Hilbert transformation	5
$\mathrm{Re}\{\cdot\}$, $\mathrm{Im}\{\cdot\}$	Real and imaginary part of a complex number	5, 7

Constants and Standard Functions

Variable	*Meaning*	*Page*
j	Imaginary unit, $\mathrm{j}^2 = -1$	5
e	Euler number	5
π	The number π	6
$\emptyset$	The empty set	65
$\Gamma(\cdot)$	The Gamma function	50
$\boldsymbol{I}_i$	Identity matrix, size $i \times i$	73
$\mathrm{Q}(\cdot)$	Complementary Gaussian error integral	48
$\mathrm{ld}(\cdot)$	Logarithm dualis, $\log_2(\cdot)$	9
$\ln(\cdot)$	Logarithm naturalis, $\log_{\mathrm{e}}(\cdot)$	16
$\mathrm{sgn}(\cdot)$	Sign function	103

Bibliography

[Ado74] J.-P. Adoul, "Error intervals and cluster density in channel modeling (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 1, pp. 125–129, January 1974.

[ADP02] K. Andrews, S. Dolinar, and F. Pollara, "LDPC decoding using multiple representations," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Lausanne, Switzerland, June 2002, p. 456.

[AGW07] K. Abdel-Ghaffar and J. Weber, "Complete enumeration of stopping sets of full-rank parity-check matrices of Hamming codes," *IEEE Transactions on Information Theory*, vol. 53, no. 9, pp. 3196–3201, September 2007.

[Ana01] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proceedings of the 44th annual IEEE Global Telecommunications Conference (GlobeCom)*, San Antonio, Texas, November 2001, pp. 1021–1025.

[AS00] N. Alon and J. Spencer, *The Probabilistic Method*, ser. Interscience Series in Discrete Mathematics and Optimization. John Wiley, 2000.

[Bac79] M. Bacharach, "Normal Bayesian dialogues," *Journal of the American Statistical Association*, vol. 74, pp. 837–846, 1979.

[BCJR74] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions Information Theory*, vol. 20, pp. 284–287, March 1974.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding," in *Proceedings of the IEEE International Conference on Communications (ICC)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[BRC60] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, March 1960.

[CCSV08] S. Chilappagari, M. Chertkov, M. G. Stepanov, and B. Vasic, "Instanton-based techniques for analysis and reduction of error floors of LDPC codes," *submitted to IEEE JSAC on Capacity Approaching Codes*, October 2008.

[CGW07] A. Casado, M. Griot, and R. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *Proceedings of the IEEE International Conference on Communications (ICC)*, Glasgow, Scotland, June 2007, pp. 932–937.

[Cha72] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170 – 182, January 1972.

[CRU01] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657–670, February 2001.

[CW90] R. Clemen and R. Winkler, "Unanimity and compromise among probability forecasters," *Management Science*, vol. 36, pp. 767–779, 1990.

[CW99] ——, "Combining probability distributions from experts in risk analysis," *Risk Analysis*, vol. 19, no. 2, pp. 187–204, 1999.

[DHLH06] A. Doenmez, T. Hehn, S. Laendner, and J. Huber, "Comparison of high-performance codes on AWGN channel with erasures," in *Proceedings of the International Symposium on Turbo Codes and Related Topics*, Munich, Germany, April 2006.

[Doh03] K. Dohmen, *Improved Bonferroni Inequalities via Abstract Tubes*. Springer-Verlag, 2003.

[DPT+02] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1570–1579, June 2002.

[EL75] P. Erdös and L. Lovász, "Problems and results on 3-chromatic hypergraphs and some related questions," in *Infinite and Finite Sets*, A. H. et al., Ed. North-Holland, Amsterdam, 1975, pp. 609–628.

[Eli55] P. Elias, "Coding for noisy channels," in *IRE National Convention Record*, ser. Part 4, 1955, pp. 37–47.

[Fel03] J. Feldman, "Decoding error-correcting codes via linear programming," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2003. [Online]. Available: http://www.columbia.edu/∼jf2189/pubs.html

[FL95] M. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, pp. 1379–1396, September 1995.

[Fos01] M. P. C. Fossorier, "Iterative reliability-based decoding of low-density parity check codes," *IEEE Journal on Selected Areas in Communication*, vol. 19, no. 5, pp. 908–917, May 2001.

[GA07] O. Gorlov and O. Amrani, "Edge-based scheduled BP in LDPC codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Nice, France, June 2007, pp. 2376–2380.

[Gal63] R. Gallager, "Low-density parity-check codes," Ph.D. dissertation, Massachusetts Institute of Technology, 1963.

[Gal68] R. G. Gallager, *Information Theory and Reliable Communication*. John Wiley and Sons, 1968.

[Gol49] M. Golay, "Notes on digital coding," in *Proceedings IEEE*, vol. 37, 1949, p. 657.

[GR99] G. Givens and P. Roback, "Logarithmic pooling of priors linked by a deterministic simulation model," *Journal of Computational and Graphical Statistics*, vol. 30, pp. 452–478, 1999.

[Gre65] T. Grettenberg, "Representation theorem for complex normal processes (corresp.)," *Transactions on Information Theory*, vol. 11, no. 2, pp. 305–306, April 1965.

[GRS01] B. Girod, R. Rabenstein, and A. Stenger, *Signals and Systems*. John Wiley & Sons, Ltd., 2001.

[HC06] T. Halford and K. Chugg, "Random redundant soft-in soft-out decoding of linear block codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Seattle, Washington, USA, July 2006, pp. 2230–2234.

[HC08] ——, "Transactions letters - random redundant iterative soft-in soft-out decoding," *IEEE Transactions on Communications*, vol. 56, no. 4, pp. 513–517, April 2008.

[HDH05a] T. Hehn, A. Doenmez, and J. Huber, "Exact thresholds for LDPC codes transmitted over binary erasure channels," in *Proceedings of the 43rd Annual Allerton Conference on Communication, Control and Computing*, Monticello, USA, September 2005, pp. 1787–1794.

[HDH05b] ——, "Introduction to the PEG algorithm," 2005, internal tutorial at the Institute of Information Transmission, University of Erlangen-Nuremberg. Slides available on request.

[He07] P. He, "Modified belief-propagation decoding," Master's thesis, University of Erlangen-Nuremberg, Erlangen, Germany, August 2007.

[HEA05] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, pp. 386–398, January 2005.

[HFE04] X.-Y. Hu, M. Fossorier, and E. Eleftheriou, "On the computation of the minimum distance of low-density parity-check codes," in *Proceedings of the IEEE International Conference on Communications (ICC)*, Paris, France, June 2004, pp. 767–771.

[HH09a] T. Hehn and J. Huber, "LDPC codes and convolutional with equal structural delay: A comparison," *IEEE Transactions on Communications*, vol. 57, no. 6, pp. 1683–1692, June 2009.

[HH09b] J. Huber and T. Hehn, "The lowest-possible BER and FER for any discrete memoryless channel with given capacity," *accepted for publication in IEEE Transactions on Communications*, March 2009.

[HHHL08] T. Hehn, J. Huber, P. He, and S. Laendner, "Multiple-bases belief-propagation with leaking for decoding of moderate-length block codes," in *Proceedings of the International ITG Conference on Source and Channel Coding (SCC)*, Ulm, Germany, January 2008.

[HHLM07] T. Hehn, J. Huber, S. Laendner, and O. Milenkovic, "Multiple-bases belief-propagation decoding for short block-codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Nice, France, June 2007, pp. 311–315.

[HHML07] T. Hehn, J. Huber, O. Milenkovic, and S. Laendner, "Multiple-bases belief-propagation decoding of high-density cyclic codes," *submitted to IEEE Transactions on Communications*, 2007.

[HHML09] ——. (2009, May) Multiple-bases belief-propagation decoding of high-density cyclic codes. Website. [Online]. Available: http://arxiv.org/abs/0905.0079

[HJ71] J. Heller and I. Jacobs, "Viterbi decoding for satellite and space communication," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 835–848, October 1971.

[HLMH06] T. Hehn, S. Laendner, O. Milenkovic, and J. B. Huber, "The stopping redundancy hierarchy of cyclic codes," in *Proceedings of the 44th Annual Allerton Conference on Communication, Control and Computing*, Monticello, USA, September 2006, pp. 1271–1280.

[HMLH07] T. Hehn, O. Milenkovic, S. Laendner, and J. Huber, "Permutation decoding and the stopping redundancy hierarchy of linear block codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Nice, France, June 2007, pp. 2926–2930.

[HMLH08] ——, "Permutation decoding and the stopping redundancy hierarchy of cyclic and extended cyclic codes," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5308–5331, December 2008.

[Hoc59] A. Hocquenghem, "Codes correcteurs d' erreurs," *Chiffres*, vol. 2, pp. 147–156, September 1959.

[Hol91] K. Hole, "Punctured convolutional codes for the 1-D partial-response channel," *IEEE Transactions on Information Theory*, vol. 37, no. 3, pp. 808–817, May 1991.

[HS07] J. Han and P. Siegel, "Improved upper bounds on stopping redundancy," *IEEE Transactions on Information Theory*, vol. 53, no. 1, pp. 90–104, January 2007.

[HT07] H. Hollmann and L. Tolhuizen, "On parity-check collections for iterative erasure decoding that correct all correctable erasure patterns of a given size," *IEEE Transactions on Information Theory*, vol. 53, no. 2, pp. 823–828, February 2007.

[Hub82] J. B. Huber, "Codierung fuer gedaechtnisbehaftete Kanaele (in German)," Ph.D. dissertation, Hochschule der Bundeswehr Muenchen, Munich, Germany, April 1982.

[Hub92] J. Huber, *Trelliscodierung (in German)*, ser. Nachrichtentechnik. Erlangen, Germany: Springer-Verlag Berlin Heidelberg, 1992.

[Hub02] J. B. Huber, "Grundlagen der wahrscheinlichkeitsrechnung für iterative decodierverfahren (in German)," *E&I (Elektrotechnik und Informationstechnik)*, vol. 119, no. 11, pp. 386–394, November 2002.

[Hub06] ——, *Nachrichtenuebertragung (in German).* Lecture Notes at the Institute for Information Transmission, 2006.

[Hub07] ——, *Information Theory and its Application to Communications Engineering.* Erlangen, Germany: Lecture Notes at the Institute for Information Transmission, 2007.

[Hue04] S. Huettinger, "Analysis and design of power-efficient coding schemes," Ph.D. dissertation, University of Erlangen-Nuremberg, Erlangen, Germany, 2004.

[IEE05] IEEE Std. 802.16e, "IEEE standard for local and metropolitan area networks, Part 16: Air interface for fixed and mobile broadband wireless access systems. Amendment 2: Physical and medium access control layers for combined fixed and mobile operation in licensed bands," Institute of Electrical and Electronics Engineers (IEEE), Tech. Rep., December 2005.

[IEE06] IEEE Std. 802.3an, "IEEE Standard for information technology, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications," Institute of Electrical and Electronics Engineers, Tech. Rep., June 2006.

[II97] M. Isaka and H. Imai, "Convolutional codes for partial response channels," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Ulm, Germany, July 1997, p. 459.

[Jaf] D. B. Jaffe. Information about binary linear codes. Web database. [Online]. Available: http://www.math.unl.edu/∼djaffe2/codes/webcodes/codeform.html

[JN06] J. Jiang and K. Narayanan, "Iterative soft-input soft-output decoding of Reed-Solomon codes by adapting the parity-check matrix," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3746–3756, August 2006.

[JTC05] JTC Joint Technical Committee, "Digital video broadcasting (DVB); second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications," European Telecommunications Standards Institute, Tech. Rep., March 2005.

[JY04] D. Johnston and H. Yaghoobi. (2004, January) Peering into the WiMAX spec: Part 1. [Online]. Available: http://www.commsdesign.com/showArticle.jhtml ?articleID=17500156

[JZ99] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding.* IEEE Press, 1999.

[Key] J. Key. Some recent developments in permutation decoding. Presentation. [Online]. Available: http://www.ces.clemson.edu/∼keyj/Key/Cork06.pdf

[KFL01] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[KS07] C. Kelley and D. Sridhara, "Pseudocodewords of Tanner graphs," *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 4013–4038, November 2007.

[KTJF05] A. Kothiyal, O. Y. Takeshita, W. Jin, and M. Fossorier, "Iterative reliability-based decoding of linear block codes with adaptive belief propagation," *IEEE Communications Letters*, vol. 9, no. 12, pp. 1067–1069, December 2005.

[Kwe75] S. Kwerel, "Most stringent bounds on aggregated probabilities of partially specified dependent probability systems," *Journal of the American Statistical Association*, vol. 70, pp. 472–479, 1975.

[Lae09] S. Laendner, "Improving the error floor performance of iteratively decodable codes," Ph.D. dissertation, University of Erlangen-Nuremberg, Germany, to appear 2009.

[LBB98] R. Lucas, M. Bossert, and M. Breitbach, "On iterative soft-decision decoding of linear binary block codes and product codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 276–296, February 1998.

[LC04] S. Lin and D. Costello, *Error Control Coding*, 2nd ed. Pearson Education, Inc., 2004.

[Lec03] G. Lechner, "Convergence of the sum-product algorithm for short low-density parity-check codes," Master's thesis, Vienna University of Technology, Vienna, Austria, April 2003.

[LH06] I. Land and J. Huber, "Information combining," *Foundations and Trends in Communications and Information Theory*, vol. 3, no. 3, pp. 227–330, November 2006.

[LHMH06] S. Laendner, T. Hehn, O. Milenkovic, and J. Huber, "When does one redundant parity-check equation matter?" in *Proceedings of the 49th annual IEEE Global Telecommunications Conference (GlobeCom)*, San Francisco, USA, November 2006.

[LM05] S. Laendner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," in *Proceedings of the International Conference on Wireless Networks, Communications, and Mobile Computing (WirelessComm)*, Maui, Hawaii, June 2005, pp. 630–635.

[LMSS98] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation," in *Proceedings of the International Symposium on Information Theory*, Cambridge, UK, August 1998, p. 117.

[Mac99] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, March 1999.

[Mac03] ——, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[MB01a] Y. Mao and A. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," in *Proceedings of the IEEE International Conference on Communications (ICC)*, vol. 1, Helsinki, Finland, June 2001, pp. 41–44.

[MB01b] ——, "A new schedule for decoding low-density parity-check codes," in *Proceedings of the 44th annual IEEE Global Telecommunications Conference (GlobeCom)*, San Antonio, Texas, USA, November 2001.

[Mea55] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, pp. 1054–1079, September 1955.

[Mil06] O. Milenkovic, "Private communication with J. Han and P. Siegel," 2006.

[MN96] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low-density parity-check codes," *Electronic Letters*, vol. 32, pp. 1645–1646, 1996.

[MS77] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland Publishing Company, 1977.

[Nea] R. M. Neal. Software for Low Density Parity Check Codes. Website. [Online]. Available: http://www.cs.toronto.edu/∼radford/ftp/LDPC-2001-11-18/index.html

[OVZ03] A. Orlitsky, K. Viswanathan, and J. Zhang, "Stopping set distribution of LDPC code ensembles," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Yokohama, Japan, July 2003, p. 123.

[Pee98] S. Peeters, "Vergleich von Kanalcodierverfahren," Studienarbeit (in German), University of Erlangen-Nuremberg, Erlangen, Germany, August 1998.

[Pro08] J. G. Proakis, *Digital Communications*, 5th ed. McGraw-Hill Publishing Company, January 2008.

[RSU01] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, February 2001.

[RU01] T. Richardson and R. Urbanke, "The capacity of LDPC codes under message passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, February 2001.

[RU08] ——, *Modern Coding Theory*. Cambridge University Press, January 2008.

[Sha48] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July and October 1948.

[Sha59] ——, "Probability of error for optimal codes in a Gaussian channel," *Bell System Technical Journal*, vol. 38, pp. 611–656, May 1959.

[SV04] N. Santhi and A. Vardy, "On the effect of parity-check weights in iterative decoding," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Chicago, Illinois, USA, July 2004, p. 322.

[SV05] M. Schwartz and A. Vardy, "On the stopping distance and stopping redundancy of codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Adelaide, Australia, September 2005, pp. 975–979.

[SV06] ——, "On the stopping distance and stopping redundancy of codes," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 922–932, March 2006.

[Tan81] M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, September 1981.

[tB99] S. ten Brink, "Convergence of iterative decoding," *IEE Electronics Letters*, vol. 35, no. 10, pp. 806–808, May 1999.

[Tre01] H. L. V. Trees, *Detection, Estimation, and Modulation Theory, Part III: Radar-Sonar Signal Processing and Gaussian Signals in Noise*. John Wiley and Sons, Inc., 2001.

[Urb] R. Urbanke. LdpcOpt - a fast and accurate degree distribution optimizer for LDPC ensembles. Website. [Online]. Available: http://lthcwww.epfl.ch/research/ldpcopt/index.php

[VF04] N. Varnica and M. Fossorier, "Belief-propagation with information correction: Improved near maximum-likelihood decoding of low-density parity-check codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, Chicago, USA, July 2004, p. 343.

[VHR+07] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS," in *IEEE international Solid-State Circuits Conference (ISSCC), Digest of Technical Papers*, San Francisco, CA, USA, February 2007, pp. 98, 99, 598.

[Vit67] A. Viterbi, "Error-bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.

[WAG05] J. Weber and K. Abdel-Ghaffar, "Stopping and dead-end set enumerators for binary Hamming codes," in *Proceedings of the Twenty-sixth Symposium on Information Theory in the Benelux*, Brussels, Belgium, May 2005, pp. 165–172.

[Wei02] E. W. Weisstein, *CRC Concise Encyclopedia of Mathematics Second Edition*. Chapman & Hall/CRC, 2002.

[Wie07] G. Wiechman, "Private communication," April 2007, by E-mail.

[Win68] R. Winkler, "The consensus of subjective probability distributions," *Management Science*, vol. 15, pp. 361–375, 1968.

[Wol83] J. Wolfmann, "A permutation decoding of the (24,12,8) golay code," *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 748–750, September 1983.

[WS07] G. Wiechman and I. Sason, "An improved sphere-packing bound for finite-length codes on symmetric memoryless channels," *Submitted to IEEE Transactions on Information Theory*, March 2007.

[XB04] H. Xiao and A. H. Banihashemi, "Improved progressive-edge-growth (PEG) construction of irregular LDPC codes," *IEEE Communications Letters*, vol. 8, no. 12, pp. 715–717, December 2004.

[ZWFY07] J. Zhang, Y. Wang, M. Fossorier, and J. Yedidia, "Iterative decoding with replicas," *IEEE Transactions on Information Theory*, vol. 53, no. 5, pp. 1644–1663, May 2007.