Technische Universität München
Lehrstuhl für Nachrichtentechnik
Prof. Dr. sc. techn. Gerhard Kramer

Master's Thesis

Analysis and Optimization of the Chase-Pyndiah Decoding Algorithm

Vorgelegt von:
Fabian Graf

München, Juni 2021

Betreut von:
Dr. Gianluigi Liva
Prof. Alexandre Graell i Amat
Juan Diego Lentner Ibañez, M.Sc.

Master's Thesis am
Lehrstuhl für Nachrichtentechnik (LNT)
der Technischen Universität München (TUM)
Titel : Analysis and Optimization of the Chase-Pyndiah Decoding Algorithm
Autor : Fabian Graf

ga47tey@mytum.de

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

München, June 4, 2021

Ort, Datum (Fabian Graf)

Contents

List of Figures

List of Tables

Abstract

Optical communication systems operate at very high throughputs and need to achieve low error rates at the same time. In order to meet both requirements, the Chase-Pyndiah (Ch-Py) algorithm for product codes is implemented in many state of the art applications, because it offers an excellent trade-off between performance and complexity. Two different strategies on optimizing this algorithm in terms of bit error rate (BER) are proposed, while keeping the complexity at a low level. One of these approaches is also used in the computation of iterative decoding thresholds for Ch-Py decoding of product codes. The thresholds are based on asymptotic analysis of the corresponding generalized low density parity check (GLDPC) code ensemble under the assumption of infinitely large blocklengths.

1. Introduction

In a time where we experience a growing demand for digital communications, probably most of the people are not aware of the fact that their exchanged messages are converted into bitstreams, which are transmitted through fiber-optic links at some point. Due to the small loss compared to other communication mediums, optical fibers are especially used for transmission over long distances with very high data rates [1].
Originally, simple uncoded on-off keying modulation was used in optical communications, but with technical progress the possibility of using forward error correction (FEC) schemes became possible [2]. The data rates increased substantially and current research has the goal to satisfy the requirements of optical systems operating at throughputs around 1 Tbit/s.
The design of channel codes with very large blocklengths turns out to be the solution to this challenging problem [3]. Especially advantageous constructions such as product codes, which have a matrix structure consisting of smaller component codes, offer a very good trade-off between performance and complexity [4]. Product codes can be decoded iteratively in a hard decision (HD) and soft decision (SD) fashion. Decoders which employ HD decoding are able to cope with very high data rates but product codes can exploit their full potential when SD decoding is applied. Nevertheless, good performances then come at the expense of significantly larger decoding complexity and high internal data flow resulting in a larger power consumption. In 1998 Ramesh Pyndiah presented a sub-optimal SD decoding algorithm of product codes, which can be efficiently implemented with low complexity and is therefore used in numerous state of the art communication systems [5]. In literature this algorithm is usually referred to as Chase-Pyndiah (Ch-Py) decoding.
In this thesis, the Ch-Py algorithm is analyzed in detail and some curiosities are highlighted. Due to its suboptimal nature, soft information generated by a Ch-Py decoder is not based on a true maximum-a-posteriori (MAP) estimation but just an approximation. It turns out that scaling this soft output is essential to achieve good FEC performance. Based on heuristic search, Pyndiah delivered such scaling coefficients in [5]. We will present two different strategies to derive optimized parameters with the goal to improve the BER performance without a large increase in additional computational complexity.

Furthermore, the asymptotic decoding behavior for the Ch-Py algorithm under the assumption of infinitely large blocklength is analyzed in this thesis. Since the original decoding algorithm violates the principle of only exchanging extrinsic information in the iterative process, a new extrinsic version of the algorithm is introduced. The extrinsic version comes with a huge increase in complexity and has the only purpose to make density evolution (DE) analysis possible. As underlying idea, the concept of generalized low density parity check (GLDPC) codes is used, where product codes are an instance of. The code class of GLDPC codes offers the feature of performing DE, where the iterative decoding threshold of the whole GLDPC code ensemble can be determined.
The remainder of this thesis is structured as follows. In Chapter 2 fundamental bounds and limits in a communication system are derived based on quantities from information theory. In addition, basic ideas of channel coding are presented, which is the tool to approach these limits practically. The structure of product codes and the Ch-Py algorithm are introduced in Chapter 3. Chapter 4 deals with two different strategies on optimizing the parameters of the Ch-Py algorithm. In Chapter 5 a new DE approach for Ch-Py decoding based on a Monte-Carlo (MC) method is proposed and the thresholds are computed. Finally the key results are summarized in Chapter 6 and an outlook on further research tendencies concerning Ch-Py decoding is provided.

2. Preliminaries

Before diving into the field of *modern coding theory*, we first want to build a framework based on information theory and *classical coding theory*. In 1948 Shannon published his seminal paper "A Mathematical Theory of Communication" [6]. Since that time the problem of finding practical coding systems that approach the fundamental limits established by Shannon, has been at the heart of information theory and communications. According to Shannon, the core problem of communication can be reduced to an abstract point-to-point transmission scenario, where a transmitted message, chosen from a set of possible messages, has to be reproduced at the receiver's side. With the help of this level of abstraction, Shannon introduced *entropy* as the measure of uncertainty about the received message. Based on entropy, the derivation of bounds on minimal representation of information (source coding theorem) and on information transmission (channel coding theorem) was possible.
In this chapter, we start with introducing some useful information theoretic quantities in Section 2.1. They will find application in the following Section 2.2, where we present two basic binary models and their corresponding capacities. In literature exist many different definitions of signal to noise ratio (SNR) and therefore it is essential to agree on a definition in Section 2.3, which is consistently used throughout this thesis. In Section 2.4 of this chapter fundamental limits for the transmission over the binary input additive white Gaussian noise (bi-AWGN) channel model are derived. In Section 2.5 basic channel coding concepts are presented, which are the key tool to approach these limits. The chapter concludes with Section 2.6 where Bose-Chaudhuri-Hocquenghem (BCH) codes as a special member of the family of cyclic codes are introduced.

2.1. Basics in Information Theory

In [6, Sec. 6] Shannon has brought up the question of how much information is "produced" by the choice from a set of equally likely messages. He defined the logarithm of the number of elements in the message set as a measure of the produced information when a message is chosen from this set. For the case of non-equally likely choices, Shannon later developed from this idea the measure *entropy* or uncertainty. The following fundamental definitions

are an excerpt of quantities from [7] and necessary for different derivations throughout this thesis.

Consider a discrete random variable X with alphabet $\mathcal{X}$ and probability distribution $P_X(x)$. Furthermore, let $\mathrm{supp}(P_X)$ be the support of the distribution P_X, i.e., the set of a such that $P_X(a) > 0$. Then, the entropy $\mathrm{H}(X)$ of the discrete random variable X is defined as

$$\mathrm{H}(X) = \sum_{a \in \mathrm{supp}(P_X)} -P_X(a) \log_2 P_X(a). \tag{2.1}$$

The entropy of a random variable is measured in bits and in general bounded by

$$0 \leq \mathrm{H}(X) \leq \log_2 |\mathcal{X}|. \tag{2.2}$$

Equality on the left is achieved if and only if there is one letter $a \in \mathcal{X}$ with $P_X(a) = 1$, and equality on the right is fulfilled if and only if X is uniform over $\mathcal{X}$, i.e. $P_X(a) = 1/|\mathcal{X}|$ for all $a \in \mathcal{X}$.

For the special case of a Bernoulli distribution with $P_X(1) = 1 - P_X(0) = p$, the entropy formula can be simplified to the binary entropy function

$$\mathrm{H}_2(p) = -p \log_2 p - (1-p) \log_2 (1-p). \tag{2.3}$$

We can extend the measure of entropy on two or more random variables, which form a conditional or joint distribution, which we denote by $P_{X|Y}$ and P_{XY}, respectively.

The conditional entropy of X given the event $Y = b$ with probability $\Pr[Y = b] > 0$ is defined as

$$\mathrm{H}(X|Y=b) = \sum_{a \in \mathrm{supp}\left(P_{X|Y}(\cdot|b)\right)} -P_{X|Y}(a|b) \log_2 P_{X|Y}(a|b). \tag{2.4}$$

The conditional entropy of X given Y can be obtained by averaging over all $b \in \mathcal{Y}$ and is defined as

$$\mathrm{H}(X|Y) = \sum_{b \in \mathrm{supp}(P_Y)} P_Y(b) \mathrm{H}(X|Y=b). \tag{2.5}$$

If and only if the random variable X is essentially determined by Y, i.e., for every b in $\mathrm{supp}\,(P_Y)$ there is an a such that $P_{X|Y}(a|b) = 1$, then the lower bound $\mathrm{H}(X|Y) = 0$ is achieved because there is no uncertainty about X if Y is known. Only for statistically independent random variables X and Y the upper bound $\mathrm{H}(X|Y) \leq \mathrm{H}(X)$ is met with equality, i.e., conditioning does not increase entropy.

The entropy of the joint random variable XY following the distribution P_{XY} can be

calculated as

$$\mathrm{H}(XY) = \sum_{(a,b)\in\mathrm{supp}(P_{XY})} -P_{XY}(a,b)\log_2 P_{XY}(a,b). \tag{2.6}$$

When applying Bayes' rule, we can find a way of expressing the joint entropy in terms of the entropy of a single random variable and the conditional entropy.

$$\mathrm{H}(XY) = \mathrm{H}(X) + \mathrm{H}(X|Y) = \mathrm{H}(Y) + \mathrm{H}(Y|X). \tag{2.7}$$

The idea of expansion of the joint entropy as in (2.7) can be generalized for more than two random variables. By defining a sequence of n random variables as $X^n = X_1X_2\ldots X_n$ and treating X^0 as constant, the formulation of the *chain rule* of entropy can be obtained:

$$\begin{aligned}\mathrm{H}\left(X_1X_2\ldots X_n\right) &= \mathrm{H}\left(X_1\right) + \mathrm{H}\left(X_2|X_1\right) + \ldots + \mathrm{H}\left(X_n|X_1X_2\ldots X_{n-1}\right)\\ &= \sum_{i=1}^{n}\mathrm{H}\left(X_i|X^{i-1}\right).\end{aligned} \tag{2.8}$$

In practical applications like the channel models, the *mutual information*, which measures the dependency of two random variables X and Y, is of great importance. Mutual information is defined as

$$\mathrm{I}(X;Y) = \sum_{(a,b)\in\mathrm{supp}(P_{XY})} P_{XY}(a,b)\log_2\frac{P_{XY}(a,b)}{P_X(a)P_Y(b)} \tag{2.9}$$

and for statistical independent random variables X and Y, i.e., $P_{XY} = P_XP_Y$, we get $\mathrm{I}(X;Y) = 0$. We further have

$$\begin{aligned}\mathrm{I}(X;Y) &= \mathrm{H}(Y) - \mathrm{H}(Y|X)\\ &= \mathrm{H}(X) - \mathrm{H}(X|Y)\\ &= \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(XY).\end{aligned} \tag{2.10}$$

For continuous random variables with a probability density function (PDF) p_X, *differential entropy* is analogously defined as

$$h(X) = \int_{\mathrm{supp}(p_X)} -p_X(a)\log_2 p_X(a)\mathrm{d}a \tag{2.11}$$

and the corresponding expressions for mutual information from (2.10) hold equivalently.

2.2. Channel Models

A channel is defined as part of a communication system, which distorts the transmitted signal [8, Sec. 1]. Throughout this thesis, we restrict ourselves to channels with a binary input. The communication rate R over a channel is defined as the fraction of correctly transmitted bits over the number of channel uses, i.e.,

$$R = \frac{\text{number of correctly transmitted bits}}{\text{number of channel uses}}. \tag{2.12}$$

The capacity C is defined as the maximum achievable rate at which reliable communication is still possible and can be expressed in terms of mutual information. Hence, for channels with no cost constraint in terms of power or energy, the rate

$$C = \max_{P_X} \mathrm{I}(X;Y) \tag{2.13}$$

can be approached by optimizing the input distribution P_X [7, Sec. 5.5].

2.2.1. Binary Symmetric Channel

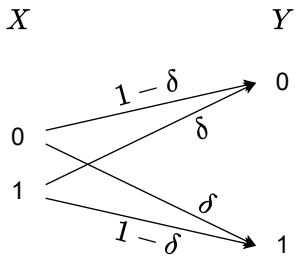


Figure 2.1.: Binary symmetric channel model.

At first, we introduce the most trivial version of a channel called binary symmetric channel (BSC) and depicted in Figure 2.1. It is characterized by a binary input X, a binary output Y and the channel *crossover* probability δ, i.e., the probability that the BSC erroneously flips the input bit during a transmission. The BSC is *symmetric* and therefore the entropy of the output Y, conditioned on the input X, does not depend on the distribution P_X of the channel input and it holds that $\mathrm{H}(Y|X) = \mathrm{H}(Y|X = x)$. To obtain the BSC capacity, we start by rewriting $\mathrm{I}(X;Y)$ in (2.13) in terms of entropy and using the binary entropy

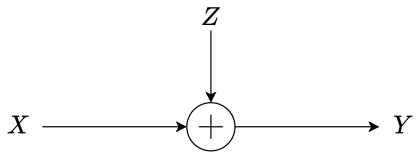


Figure 2.2.: Additive white Gaussian noise channel model.

function $\mathrm{H}_2(\cdot)$ from (2.3). According to [7, Sec. 5.7] we get

$$\begin{aligned} \mathrm{I}(X;Y) &= \mathrm{H}(Y) - \mathrm{H}(Y|X) \\ &= \mathrm{H}(Y) - \sum_{x\in\{0,1\}} P_X(x)\mathrm{H}(Y|X=x) \\ &= \mathrm{H}(Y) - \sum_{x\in\{0,1\}} P_X(x)\mathrm{H}_2(\delta). \end{aligned} \tag{2.14}$$

The expression above is maximized when choosing a uniformly distributed input, i.e., $P_X(0) = P_X(1) = 0.5$. Thus, the output Y is obviously also uniform and the capacity of the BSC is

$$C_{\mathrm{BSC}} = 1 - \mathrm{H}_2(\delta). \tag{2.15}$$

2.2.2. Additive White Gaussian Noise Channel

In this subsection the time-discrete additive white Gaussian noise (AWGN) channel is introduced. It is depicted in Figure 2.2 and is assumed to be a memoryless channel, so the output Y, conditioned on X, is independent of the former time instances. The channel output Y is defined as the sum of channel input X and noise Z. The noise samples are drawn from a Gaussian distribution with zero mean and variance σ^2 and thus we may write

$$Y = X + Z\ , \quad Z \sim \mathcal{N}(0, \sigma^2). \tag{2.16}$$

The channel transition probability is given by the conditional PDF

$$p_{Y|X}(y|x) = p_Z(y-x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y-x)^2}. \tag{2.17}$$

In the following, we introduce a power constraint on the channel input X. To minimize the probability of error, it is intuitive to place the signal points infinitely far away from each other. This is not feasible in practice of course, because of the direct dependency of the modulation points and the power consumption of the communications system.
We usually measure the *power* of a signal point x by its squared value x^2 [8, Sec. 2.2]. Especially in a channel coding environment, where blocks of bits with a certain length n

are transmitted, a common less restrictive constraint is given by the *average power*

$$\frac{\mathrm{E}\left[\sum_{i=1}^{n} X_i^2\right]}{n} \leq P. \tag{2.18}$$

Thus, we can define the AWGN channel capacity in this case as

$$C_{\mathrm{AWGN}}(P) = \max_{p_X:\mathrm{E}[X^2]\leq P} \mathrm{I}(X;Y). \tag{2.19}$$

When we rewrite this mutual information in terms of entropy and use the logarithmic expression for Gaussian random variables [7, Sec. 2.3.3], we come to the following result:

$$\begin{aligned} C_{\mathrm{AWGN}}(P) &= \max_{P_X:E|X^2|\leq P} [h(Y) - h(Y|X)] \\ &= \left[\max_{P_X:E[X^2]\leq P} h(Y)\right] - \frac{1}{2}\log(2\pi e\sigma^2) \\ &= \frac{1}{2}\log(2\pi e(P+\sigma^2)) - \frac{1}{2}\log(2\pi e\sigma^2) \\ &= \frac{1}{2}\log\left(1+\frac{P}{\sigma^2}\right) \end{aligned} \tag{2.20}$$

The capacity $C(P)$ can be achieved by choosing X to be Gaussian distributed with zero mean and variance P [7, Sec. 5.8].

2.3. Signal to Noise Ratio

Assume the power constraint (2.18) is met with equality, then we can define the signal to noise ratio (SNR) of the channel as

$$\mathrm{SNR} = \frac{\mathrm{E}[X^2]}{\sigma^2} = \frac{P}{\sigma^2}. \tag{2.21}$$

In the next step we define the *energy* per information bit E_b as the power divided by the number of information bits per channel use (bpcu), which is the unit of the rate R. We also consider the fact now, that signal points may lie in the complex plane and the noise variance has to be measured in two dimensions as $2\sigma^2$. Denoting by N_0 the single-sided noise power spectral density, we can come up with another widely used definition of a SNR

$$\frac{E_b}{N_0} = \frac{\mathrm{SNR}}{2R}. \tag{2.22}$$

2.4. Limits of a bi-AWGN Communication System

As explained in Subsection 2.2.2, the AWGN channel capacity can be achieved by a continuous Gaussian input X, which is typically not feasible in practical systems. Therefore, we will restrict ourselves to the binary input additive white Gaussian noise (bi-AWGN) channel throughout the remainder of the thesis. X is constrained to the binary phase shift keying (BPSK) modulation symbol input alphabet $\mathcal{X} = \{-1, +1\}$. The mapping between codeword bits $c \in \{0, 1\}$ and x is defined as

$$x = 1 - 2c. \tag{2.23}$$

In this section we want to highlight some limits in terms of achievable rates and BERs over the bi-AWGN channel based on information theoretic results. Those limits serve as benchmarks for all presented applications in this thesis.

2.4.1. Communication Rate Limits

We follow the derivations in [9, Sec. 1.5.1.3] for the calculation of the bi-AWGN channel capacity and can observe that the maximization of the channel output entropy $h(y)$ changes due to dealing with a constrained input. By definition of differential entropy in (2.11) we can formulate the objective function

$$h(Y) = \int_{\mathrm{supp}(p_Y)} -p_Y(a) \log_2 p_Y(a) \mathrm{d}a. \tag{2.24}$$

When signal points are drawn from a finite set, such as $\mathcal{X} = \{-1, +1\}$, the resulting optimization problem can be expressed as

$$C_{\text{bi-AWGN}}(P) = \left[\max_{p_X} \int_{\mathrm{supp}(p_Y)} -p_Y(a) \log p_Y(a) \mathrm{d}a \right] - \frac{1}{2} \log(2\pi e \sigma^2), \tag{2.25}$$

$$\text{where} \quad p_Y(y) = \sum_{x \in \mathcal{X}} p_X(x) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(y-x)^2}{2\sigma^2} \right\} \tag{2.26}$$

and where choosing the signal constellation points uniformly turns out to be optimal for the bi-AWGN channel.

Many practical systems do not offer the computationally expensive possibility of detecting a SD at the channel output, but only a HD, which converts the bi-AWGN channel into a BSC with error probability

$$\epsilon = Q(\sqrt{\mathrm{SNR}}), \tag{2.27}$$

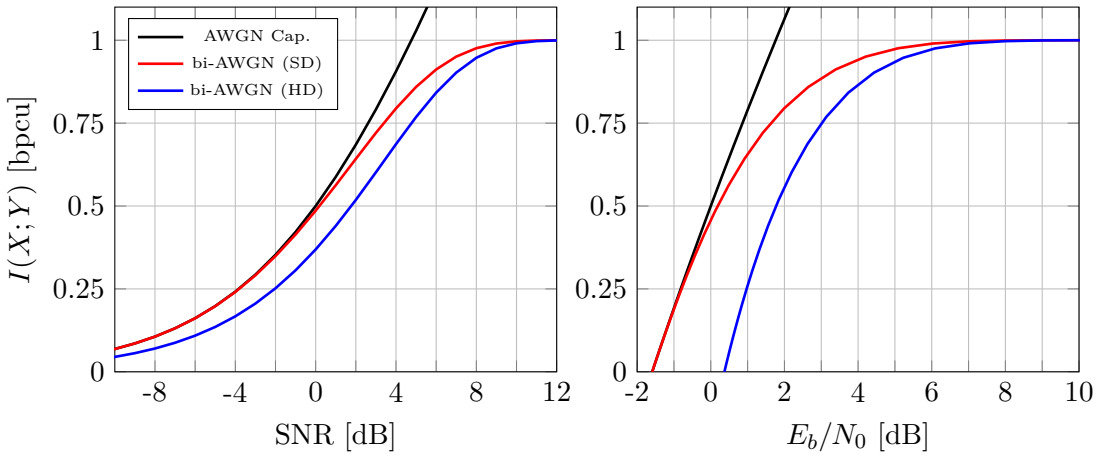


Figure 2.3.: Achievable rates for the soft-decision and hard-decision bi-AWGN channel.

where we use the Q-function defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right) du. \tag{2.28}$$

For the HD bi-AWGN channel, the achievable rates can thus be computed via the BSC capacity by

$$C_{\text{bi-AWGN (HD)}} = 1 - \mathrm{H}_2(\epsilon). \tag{2.29}$$

Rate curves for both, SD and HD, are shown in Figure 2.3. Furthermore, the capacity of an unconstrained input AWGN channel (2.20) is included. For lower rates the SD bi-AWGN capacity curve is close to the unconstrained one, but for increasing SNR the bi-AWGN capacity saturates at 1 bpcu. Higher order modulation is necessary to achieve larger rates with the same SNR. Observe that the smallest E_b/N_0 as $R \to 0$ is $E_b/N_0 = \ln(2)$, which is $(E_b/N_0)_{\text{dB}} = 10\log_{10}(\ln 2) \approx -1.6$ dB. This value indicates the ultimate minimum energy per information bit required to transmit reliably over an AWGN channel.

2.4.2. Bit Error Rate Limits

The derived limits in the previous subsection show what minimum SNR is necessary to communicate at a certain rate without any errors. However, we may allow a small fraction of errors. According to [10, Sec. 4.3], for an uncoded bi-AWGN system, the probability of receiving an erroneous symbol/ bit is given by

$$P_b = Q\left(\sqrt{\mathrm{SNR}}\right) = Q\left(\sqrt{2\frac{E_b}{N_0}}\right). \tag{2.30}$$

The derived expressions give the BER for uncoded transmission, i.e., we send the bits just like the source emits them. But there are source and channel coding strategies, which enable to achieve the same error rates with less transmit power. In the following we will use [9, Sec. 1.5.1.3] again to derive the SNR limits, which are at least necessary to communicate with a certain bit error probability P_b, and separate that the source and channel coding task, respectively. We assume an error-free channel-coding system (encoder/channel/decoder) at rate R_c and a source coding system (source encoder/source decoder) with rate R_s introducing errors at a desired rate P_b. The theoretical (lower) limit on R_s with error rate P_b is known to be

$$R_s = 1 - \mathrm{H}_2(P_b). \tag{2.31}$$

The overall rate of the communication system is $R = R_c/R_s$. Because of the assumption of a perfect channel coding system, we take the derived achievable rates from the previous Subsection 2.4.1 for R_c and the corresponding SNR value. Thus, we may write

$$C^{-1}_{\text{bi-AWGN}}(R_c) = \mathrm{SNR}. \tag{2.32}$$

In Figure 2.4 bit error probability limits for different communication rates over the bi-AWGN channel are plotted versus E_b/N_0. On top of that, the BER of an uncoded system (2.30) is given. The corresponding HD limits are plotted as dashed lines. The HD bit error rate limits can be obtained in the same fashion as described for the SD case.

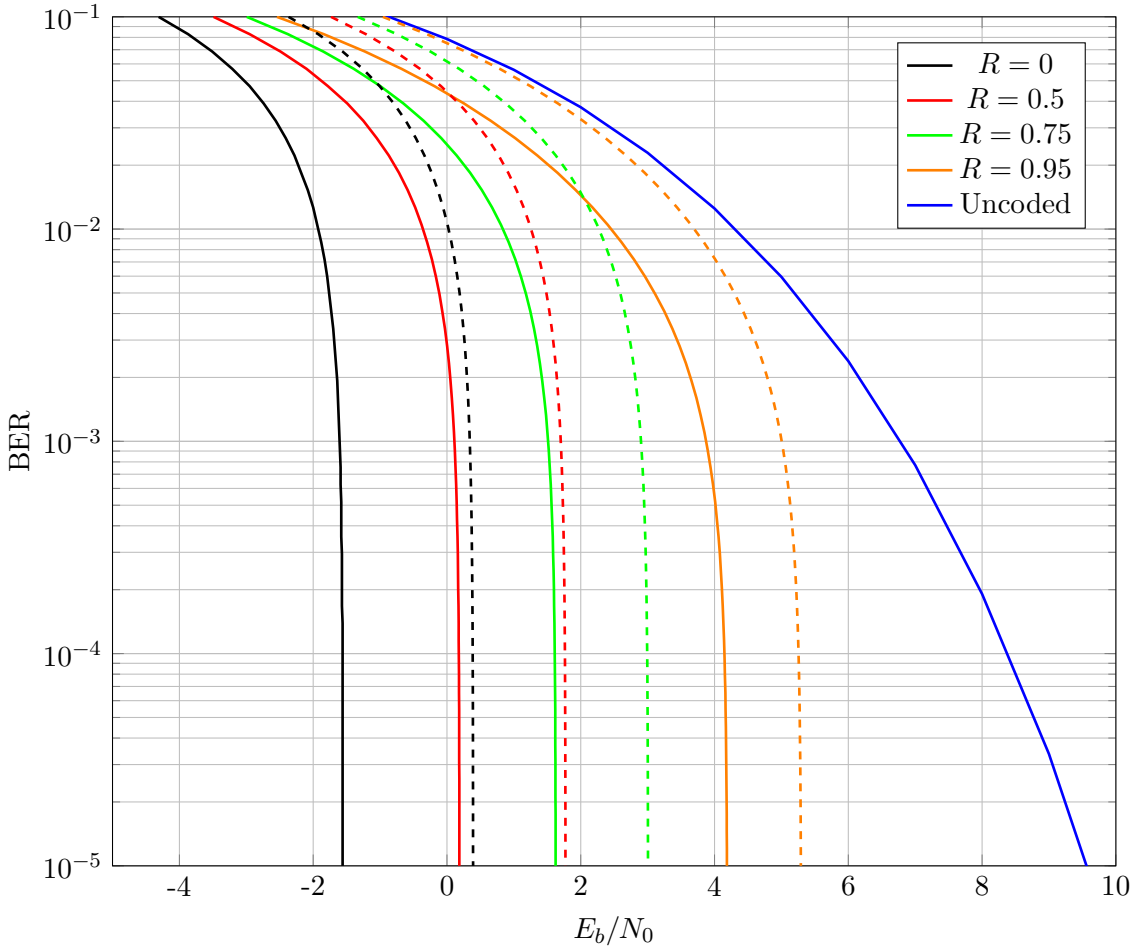


Figure 2.4.: BER limits for different communication rates over the bi-AWGN channel. The solid lines indicate the limit for SD decoding and the dashed lines for HD decoding, respectively.

2.5. Channel Coding

An intuitive way to approach the derived limits in the previous subsection is to decide for a trade-off between information rate and BER by adding redundancy to a message of k information bits. At the receiver's side, this redundancy can be used again to detect or/ and resolve possible errors caused by the noisy channel. This described process is known as *channel coding*, for which we will introduce some basic concepts in this section based on [11] as well as on standard textbooks [10, 9, 12].
The output of the channel encoder is called the codeword $\boldsymbol{c}$ with length n. Such a codeword is typically sent in n channel uses over the desired bi-AWGN channel. The definition of the code rate is thus

$$R = \frac{\text{number of information bits}}{\text{number of channel uses}} = \frac{k}{n}. \tag{2.33}$$

In this thesis we restrict ourselves to binary linear block codes $\mathcal{C}$, which are defined as a k-dimensional subspace of the vector space $\mathbb{F}_2^n$ with minimum Hamming distance d. Linear block codes have the beneficial property that the linear combination of any two codewords is again a codeword. Furthermore, a linear code always contains the all-zero codeword, denoted by $\mathbf{0} := (0, \ldots, 0)$.
A linear block code is usually represented by its generator matrix $\boldsymbol{G}$ and parity check matrix $\boldsymbol{H}$. The codebook of $\mathcal{C}$ has cardinality $|\mathcal{C}| = 2^k$ and can be obtained by multiplying all possible combinations of an information bit vector $\boldsymbol{u}$ of length k with $\boldsymbol{G}$. This is at the same time the definition of *encoding* with a linear block code.

$$\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G} \tag{2.34}$$

We call a generator matrix $\boldsymbol{G}$ *systematic* if it has the form $(\boldsymbol{I}_k|\boldsymbol{A})$, where $\boldsymbol{I}_k$ is the identity matrix of size k. If this condition is fulfilled, the parity check matrix $\boldsymbol{H}$ can easily be derived as $(-\boldsymbol{A}^T|\boldsymbol{I}_{n-k})$. To check if $\boldsymbol{c}$ is a valid codeword of $\mathcal{C}$, the condition

$$\boldsymbol{c} \cdot \boldsymbol{H}^T = \mathbf{0} \tag{2.35}$$

must be fulfilled.
Next, we define the concept of the *dual code*. We denote by $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ the scalar product between $\boldsymbol{a}$ and $\boldsymbol{b}$, which is calculated as $\sum_{i=1}^{n} a_i \cdot b_i$. Now let $\mathcal{C}$ be a linear (n, k, d) code. Then the dual code $\mathcal{C}^\perp$ is defined as

$$\mathcal{C}^\perp := \{\boldsymbol{c}^\perp \in \mathbb{F}_q^n : \langle \boldsymbol{c}^\perp, \boldsymbol{c} \rangle = 0 \quad \forall \boldsymbol{c} \in \mathcal{C}\}. \tag{2.36}$$

The dual code has parameters $(n, n-k, d^{\perp})$ and $\boldsymbol{H}$ is the generator matrix of the dual code $\mathcal{C}^{\perp}$. Note that for some code classes there is a direct connection between the minimum distance d of $\mathcal{C}$ and the minimum distance $d^{\perp}$ of the dual code $\mathcal{C}^{\perp}$, but $d^{\perp}$ is not necessarily determined by the minimum distance of $\mathcal{C}$.
Assume we transmit a codeword $\boldsymbol{c} \in \mathcal{C}$ over a BSC with cross-over probability $\delta < 0.5$ and receive the binary vector $\boldsymbol{r} \notin \mathcal{C}$. According to (2.35) the product $\boldsymbol{r} \cdot \boldsymbol{H}^T$ is therefore non-zero. This product is usually referred to as the *syndrome* $\boldsymbol{s}$ of $\boldsymbol{r}$,

$$\boldsymbol{s} := \boldsymbol{r} \cdot \boldsymbol{H}^T. \tag{2.37}$$

With the help of syndromes, we can define a first hard decision (HD) decoding strategy for a binary linear block code. For codes with a small redundancy it is an easy task to compute a *syndrome table*. Based on the parity check matrix $\boldsymbol{H}$, a syndrome table with two columns can be generated. In one column are the error patterns $\boldsymbol{e}$ and in the other one are the corresponding syndromes $\boldsymbol{s}$. We start with building all possible error patterns of Hamming weight one, i.e., $w_{\mathrm{H}}(\boldsymbol{e}) = 1$. Then we compute the corresponding syndromes according to equation (2.37) and add it to the syndrome table if $\boldsymbol{s}$ is not already contained. After that we continue with all $\binom{n}{2}$ error patterns of weight two. This procedure is carried on for increasing error pattern weights until all $2^{(n-k)}$ syndromes are found.
After the generation of such a syndrome table, we can use it in the forward error correction (FEC) process. Assume again, we have received a binary vector $\boldsymbol{r}$ at the channel output. Then the estimated codeword $\hat{\boldsymbol{c}}$ at the channel output can be obtained by calculating the syndrome $\boldsymbol{s}$ of $\boldsymbol{r}$, look it up in the syndrome table and add the corresponding error pattern $\boldsymbol{e}$ on $\boldsymbol{r}$.

$$\hat{\boldsymbol{c}} = \boldsymbol{r} + \boldsymbol{e} \tag{2.38}$$

It turns out that up to a certain weight t, the code guarantees to correct all possible error patterns of this weight uniquely, i.e., the code has an error correction capability t. Moreover, there is a direct link between the minimum distance d of $\mathcal{C}$ and t, which can be expressed as

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor. \tag{2.39}$$

For codes with a large redundancy, it is cumbersome to construct a *full* syndrome table. A good performance-complexity trade-off is to decode a linear block code only up to t errors. This approach is called bounded distance decoding (BDD) and the number of

error patterns, i.e., the size of the syndrome table, reduces to

$$\sum_{i=1}^{t} \binom{n}{i} \tag{2.40}$$

in this case. BDD is suboptimal, since we do not exploit the entire codebook of $\mathcal{C}$. Decoding with a full syndrome table would maximize the probability of deciding for the closest codeword $\hat{\boldsymbol{c}}$, based on the received block $\boldsymbol{y}$ and given all codewords $\boldsymbol{c} \in \mathcal{C}$. The maximization of this likelihood function is defined as the *block-wise* maximum likelihood (ML) decoding criterion

$$\hat{\boldsymbol{c}} = \arg\max_{\boldsymbol{c}\in\mathcal{C}} p(\boldsymbol{y}|\boldsymbol{c}). \tag{2.41}$$

Up to now, we have always taken the hard decision (HD) of a received block. In terms of bit error rate (BER) it is advantageous to decide on a *bit-wise* basis. The bit-wise maximum-a-posteriori (MAP) decoding criterion e.g. decides for each codeword bit c_i based on the value maximizing the a-posteriori probability (APP) $p(c_i|\boldsymbol{y})$, i.e.,

$$\hat{c}_i = \arg\max_{c_i\in\{0,1\}} p(c_i|\boldsymbol{y}). \tag{2.42}$$

Famous optimal decoding algorithms, which maximize (2.42), are, e.g., the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm [13] defined on the code trellis or the Hartmann-Rudolph (HR) decoding algorithm based on the dual codebook [14].

The algebraic nature of linear block codes allows an analytical prediction of the BER and frame error rate (FER) performance under ML decoding. Whereas the bounds on BER in Subsection 2.4.2 indicate limits for block codes with infinitely large blocklenghts, finite length coding bounds provide benchmarks for codes in practical applications. In this thesis especially the union bound (UB) will be used for error floor analysis. For its detailed derivation, the reader may refer to [9, Sec. 4.6.1]. The UB is a lower bound on the FER under ML decoding for linear block codes, which is dependent on the minimum distance and the weight enumerator (WE) of a code. To derive the WE of a code, the number of occurrences of codewords with Hamming weight i in the entire codebook of $\mathcal{C}$ has to be counted. Denoting such multiplicities as A_i, the frame error probability P_F for the bi-AWGN channel can be bounded by

$$P_\mathrm{F} \leq \sum_{i=d_\mathrm{min}}^{n} A_i Q\left(\sqrt{2iR\frac{E_b}{N_0}}\right). \tag{2.43}$$

For larger codes deriving the WE is a formidable task. However, the error floor in the high SNR regime is dominated by the minimum distance $d_{\min}$ and its multiplicity A_d [15]. This property can be used for an approximation of the UB, yielding the *truncated* UB

$$P_{\mathrm{F}} \approx A_d Q\left(\sqrt{2dR\frac{E_b}{N_0}}\right). \tag{2.44}$$

2.6. Bose-Chaudhuri-Hocquenghem Codes

Among linear block codes, cyclic codes provide certain practical advantages in terms of efficiency and fast implementations as the encoding/ decoding task can be realized by means of shift registers [16, Sec. 8]. We will introduce the following definitions and derivations concerning cyclic codes according to [11, Sec. 6] and [10, Sec. 4.9] in the remainder of this section.
A code is cyclic, if any cyclic shift of a codeword is again a codeword, i.e.,

$$(c_0, c_1, \ldots, c_{n-1}) \in \mathcal{C} \Longrightarrow (c_{n-1}, c_0, \ldots, c_{n-2}) \in \mathcal{C}. \tag{2.45}$$

For cyclic codes, the polynomial description of all words frequently simplifies notations. For this purpose, we associate each vector $(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ with a polynomial $c(x) := c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1}x^{n-1} \in \mathbb{F}_2[x]$. Thus, a cyclic shift of the vector then corresponds to

$$c_{n-1} + c_0 x + \cdots + c_{n-2}x^{n-1} = x \cdot c(x) - c_{n-1} \cdot (x^n - 1) = x \cdot c(x) \mod (x^n - 1). \tag{2.46}$$

That means, a linear code is cyclic if and only if

$$c(x) \in \mathcal{C} \Longrightarrow x \cdot c(x) \bmod (x^n - 1) \in \mathcal{C}. \tag{2.47}$$

Similar to the generator and parity-check matrices, we can consider a generator polynomial and a parity-check polynomial for cyclic codes. Let $\mathcal{C}$ be a cyclic (n, k, d) code. Then there is a unique monic polynomial $g(x)$ such that for every $c(x) \in \mathbb{F}_2[x]$ of degree at most $n - 1$:

$$c(x) \in \mathcal{C} \quad \Longleftrightarrow \quad g(x) \mid c(x). \tag{2.48}$$

The codebook of $\mathcal{C}$ can be generated with the help of the defined generator polynomial $g(x)$ with degree $\deg(g(x)) = n - k$

$$\mathcal{C} = \{u(x) \cdot g(x) : u(x) \in \mathbb{F}_2[x] \text{ and } \deg(u(x)) < k\}. \tag{2.49}$$

The parity-check polynomial $h(x)$ is the monic polynomial of degree k obtained by

$$h(x) := \frac{x^n - 1}{g(x)} \tag{2.50}$$

The corresponding generator matrices $\boldsymbol{G}$ and parity check matrices $\boldsymbol{H}$ of cyclic codes can be easily constructed by placing the polynomial coefficients as

$$\boldsymbol{G} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & \\ & g_0 & g_1 & \cdots & g_{n-k} & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix}, \tag{2.51}$$

$$\boldsymbol{H} = \begin{pmatrix} h_k & h_{k-1} & \cdots & h_0 & & & \\ & h_k & h_{k-1} & \cdots & h_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & h_k & h_{k-1} & \cdots & h_0 \end{pmatrix}. \tag{2.52}$$

By looking at the codebook of the most trivial code class, the repetition code, we observe that this class falls under the group of cyclic codes. The codebook of a repetition code consists of the all-zero and the all-one codeword. As a result of that, any cyclic shift of one of the two codewords results in the same codeword again. With this prior knowledge about cyclic codes, we are now ready to introduce Bose-Chaudhuri-Hocquenghem (BCH) codes. BCH codes were discovered by Bose and Ray-Chaudhuri [17] and independently by Hocquenghem [18]. This class of codes is part of many practical implementation standards thanks to their algebraic structure which allows very efficient decoding techniques.
To derive generator polynomials for BCH codes, so-called cyclotomic cosets and minimal polynomials are needed.
Let n be and integer with greatest common divisor (gcd) $\gcd(n, q) = 1$ and let m be the smallest integers such that n divides $q^m - 1$. The cyclotomic coset C_i with respect to n is defined by

$$C_i := \left\{ i \cdot q^j \mod n, \forall j = 0, 1, \ldots, n_i - 1 \right\}, \tag{2.53}$$

where n_i is the smallest integer such that $i \cdot q^{n_i} = 1$. A cyclotomic coset has the following properties:

- Their size is at most m: $|C_i| \leq m$.
- Two cyclotomic cosets are either distinct or identical: $C_i \cap C_j = \emptyset$ or $C_i = C_j$
- $C_0 = 0$

- $\bigcup_i C_i = \{0, 1, \ldots, n-1\}$

Now let C_i be the i-th cyclotomic coset with respect to n and let α be an element of $\mathbb{F}_{2^m}$, then the minimal polynomial of α^i is defined as

$$m_i(x) = \prod_{j \in C_i} \left(x - \alpha^j\right) \tag{2.54}$$

Let $D = C_{i_1} \cup C_{i_2} \cup \cdots \cup C_{i_\ell}$ be the union of $\ell \geq 1$ distinct cyclotomic cosets with respect to n, which divides 2^s (s is called the extension factor of the binary Galois field). Let $\alpha \in \mathbb{F}_{2^s}$ be an element of order n. Then, an (n, k, d) BCH code is defined by the following generator polynomial:

$$g(x) = \prod_{i \in D} \left(x - \alpha^i\right). \tag{2.55}$$

Note that only certain choices of s and n allow to construct BCH codes, because the definition of cyclotomic cosets demands $n|(2^s - 1)$, which implies that $\gcd(n, q) = 1$. This leads to the characteristic that there exist no even-length binary BCH codes. The dimension of the BCH code is given by $k = n - |D|$, where $|D|$ is the size of the union of all distinct cyclotomic cosets with respect to n.
We restrict to the class of *primitive* BCH codes and therefore the code parameters can be simply determined by a pair of the positive integers $s \geq 3$ and $t < 2^{s-1}$, where t is the error-correcting capability of the code,

$$n = 2^s - 1, \quad n - k \leq st, \quad d \geq 2t + 1. \tag{2.56}$$

A list of generator polynomials for binary primitive BCH codes can be found in [19, Appendix A]. Although the length of a BCH code is strictly defined, different code lengths can be obtained by shortening and extending. Shortening is simply achieved with a systematic encoder by setting a certain number of information bits to zero and not transmitting them. On the other side we obtain extended BCH (eBCH) codes by adding additional single parity check(s). When constructing eBCH codes of length $n = 2^s$, the encoder consists of serial concatenation of a primitive BCH code and a single-parity check (SPC) code [20, Sec. 2.6]. The generator polynomial of eBCH code is thus the result of a multiplication of the generator polynomials of the normal BCH and SPC code, respectively. Using eBCH codes is advantageous because it gives an increase to the minimum distance of the code for a very small reduction in code rate and a negligible additional decoding complexity [5]. The cyclic structure of BCH codes allows for simple syndrome-based HD decoding. A very efficient algebraic BDD algorithm for BCH codes is the Berlekamp-Massey (BM) algorithm [21], [9, Sec. 3.3], which finds the most probable

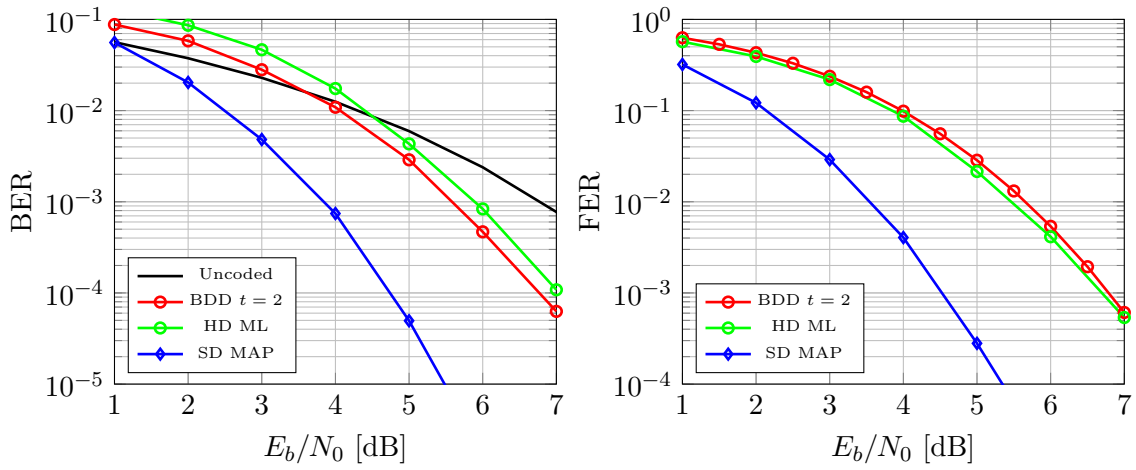


Figure 2.5.: Error performance of a (32,21,6) eBCH code for different decoding algorithms.

error pattern up to t errors introduced by the channel, iteratively given the syndrome corresponding to the received binary vector. The BM algorithm is especially useful if the number of redundancy symbols $n - k$ is very large, i.e., full syndrome table decoding is too complex. . Furthermore, the structure of the BM algorithm is suitable for efficient hardware implementations allowing very high throughputs [22].

The performance of a $(n = 32, k = 21, d = 6)$ BCH code decoded as a stand-alone linear block code simulated over a bi-AWGN channel, is shown in Figure 2.5. The black curve represents the uncoded BER (2.30). The red curve shows the error rates for the (32,21) eBCH code decoded via bounded distance decoding (BDD), which corresponds to decoding up to $t = 2$ errors, where t is the error-correcting capability of the code. If we construct a full syndrome table, we end up with HD ML decoding, which is optimal in terms of HD FER. It is remarkable how close the suboptimal but much less complex BDD algorithm performs compared to HD ML. This fact also contributes to the choice of BCH codes in numerous state of the art communication standards. The blue curve corresponds to bitwise SD MAP decoding shows the additional potential of the eBCH code when SD algorithms are applied.
The simulation results for BDD in Figure 2.5 can easily be checked because BDD offers the feature of an analytical description of FER for HD decoding. According to [10, Sec. 7.11], for a BSC with crossover probability p, the frame error probability under BDD is given by

$$P_{\mathrm{F}} = \sum_{i=t+1}^{n} \binom{n}{i} p^i \left(1-p\right)^{n-i}, \tag{2.57}$$

where n is the code length and t is the error correcting capability and the corresponding bit error probability is

$$P_b = \frac{1}{n} \sum_{i=t+1}^{n} i \binom{n}{i} p^i (1-p)^{n-i} . \tag{2.58}$$

These expressions can also be applied to the bi-AWGN channel. The channel crossover probability p can therefore be replaced by the bit error probability of an uncoded transmission over a bi-AWGN channel, which is given in (2.30) as $p = Q(\sqrt{\mathrm{SNR}})$.
At the end of this section, we want to highlight some properties about the minimum distance of BCH codes. Hamming codes are special BCH codes and have parameters $(n = 2^m - 1, k = 2^m - 1 - m, d = 3)$. Their minimum distance multiplicity is $A_d = A_3 = \binom{n}{2}\frac{1}{3}$ [23]. For extended Hamming codes the derivation of $A_d = A_4$ is studied in [24]. Finding A_d for BCH codes with $t \geq 2$ is much more complex. A solution to this problem was addressed in [25] for the first time, but we will simply use for UB computations the multiplicities from Table 2.1, which is a collection of the codes used throughout this thesis [23, 26, 27].

Code	n	k	d	t	A_d
HAM	31	26	3	1	155
eHAM	32	26	4	1	1240
BCH	31	21	5	2	186
eBCH	32	21	6	2	992
HAM	63	57	3	1	651
eHAM	64	57	4	1	10416
BCH	63	51	5	2	1890
eBCH	64	51	6	2	20160
HAM	127	120	3	1	2667
eHAM	128	120	4	1	85344
BCH	127	113	5	2	16002
eBCH	128	113	6	2	341376

Table 2.1.: Minimum distance multiplicities of some selected (e)BCH codes.

3. Product Codes

The idea of product codes was introduced by Peter Elias in 1954 [4] and later generalized by Michael Tanner in 1981 [28]. Elias' original idea was to construct a code from smaller component codes. With the invention of turbo codes [29] the principle of iterative exchange of extrinsic information about the codebits between two component decoders gave product codes the final boost in performance.
In the past 20 years the suitability of product codes for iterative decoding and their powerful error correction capability at high rates made them very attractive for several communication systems. Traditionally, low density parity check (LDPC) codes can cope with these rates and are usually decoded with the belief propagation (BP) algorithm, which requires the exchange of soft messages between the variable nodes (VNs) and check nodes (CNs) of the Tanner graph [30, 28]. Indeed, product codes can be seen as an instance of generalized low density parity check (GLDPC) codes, because they can also be represented by a bipartite graph, which performs decoding of a linear block code as a CN operation.
Product codes are currently chosen to protect the information stored in hard drives [31, 32] and to correct errors in optical fiber links operating at high data rates [33] or also in some wireless communication standards (e.g. Worldwide Interoperability for Microwave Access (WIMAX)) [34].
In this chapter we will proceed with Section 3.1 explaining the code construction of product codes. The following Section 3.2 introduces the perspective of treating product codes as GLDPC codes. The remainder of this chapter deals with the hard iterative bounded distance decoding (iBDD) and extrinsic message passing (EMP) algorithm in Section 3.3 and the soft Ch-Py turbo product decoding (TPD) algorithm in Section 3.4 [5].

3.1. Code Construction

Following Elias' idea in [4], in a 2-dimensional product code, each codeword is a 2-dimensional matrix with its rows and columns fulfilling the constraints of two binary linear blockcodes $\mathcal{C}_1$ and $\mathcal{C}_2$ with parameters (n_1, k_1, d_1) and (n_2, k_2, d_2), also called component codes. We denote by $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ the product code and define its length and dimension

by

$$n = n_1 \cdot n_2 \quad \text{and} \quad k = k_1 \cdot k_2, \tag{3.1}$$

respectively. By the definition of code rate in (2.33),the rate R of a product code can be calculated as

$$R = \frac{k}{n} = \frac{k_1}{n_1} \cdot \frac{k_2}{n_2} = R_1 \cdot R_2. \tag{3.2}$$

The k information bits are organized in a $k_2 \times k_1$ array $\mathbf{U}$. Each row of $\mathbf{U}$ is then systematically encoded via the binary linear block code $\mathcal{C}_1$. The resulting $k_2 \times n_1$ array is then systematically encoded column-wise through a binary linear block code $\mathcal{C}_2$, leading to an $n_2 \times n_1$ array $\mathbf{C}$ with the structure

$$\mathbf{C} = \left[\begin{array}{c|c} \mathbf{U} & \mathbf{P}^{(1)} \\ \hline \mathbf{P}^{(2)} & \mathbf{P}^{(1,2)} \end{array}\right]. \tag{3.3}$$

We call $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$ and $\mathbf{P}^{(1,2)}$ the parity bits for the rows, columns and the parity on parity, respectively. The shape of $\mathbf{C}$ and its dimensions are visualized in Figure 3.1.

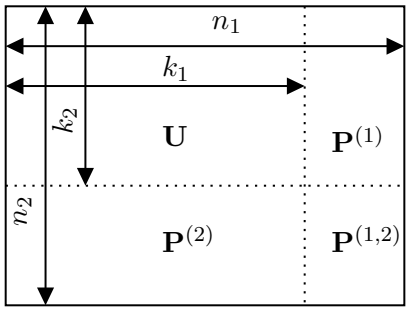


Figure 3.1.: Structure of a product code codeword and its array dimensions.

At this point it is worth mentioning that the encoding order is not relevant for the calculation of $\mathbf{P}^{(1,2)}$. The minimum distance d of the product code is

$$d = d_1 \cdot d_2. \tag{3.4}$$

Proof: Since a product code with linear component codes is also linear, it contains the all-zero codeword. Assuming that the product code has minimum distance $d = d_1 \cdot d_2$, the codeword with minimum Hamming weight has weight d. We observe that the minimum distance of the product code cannot be smaller, since any codeword of minimum weight must have at least d_1 ones in each non-zero row, and there must be at least d_2 non-zero rows. Practically, such a codeword can be constructed by selecting a weight-d_1 codeword $\mathbf{c}_1$ of $\mathcal{C}_1$ and a weight-d_2 codeword $\mathbf{c}_2$ of $\mathcal{C}_2$. By calculating the outer product, we obatin

a valid codeword $\mathbf{C} = \mathbf{c}_2^T \mathbf{c}_1$ of the product code. Since there are exactly d_1 columns equal to $\mathbf{c}_2$ and the remaining ones being all-zero, we have shown that the Hamming weight of $\mathbf{C}$ is $d_1 \cdot d_2$ [35, Sec. 5.1].
We could use the UB to investigate the FER of a product code in the error floor region. Although there is a simple connection between the parameters of the component codes and the product code, expressing the WE of the product code A_i in terms of the component code WE, i.e $A_i^{(1)}$ and $A_i^{(2)}$ turns out to be extremely complex.
However, for computations of the truncated UB, we only need A_d. It was shown in [36] that the multiplicity of codewords with minimum (non-zero) Hamming weight of a product code is equal to the product of the minimum distance multiplicities of its component codes, i.e.,

$$A_d = A_{d_1}^{(1)} A_{d_2}^{(2)}. \tag{3.5}$$

Throughout this thesis, we will restrict ourselves to two-dimensional product code constructions with the same component codes used for row and column encoding, respectively. For ease of notation, a component code with parameters (n, k, d) leads to a $(n, k, d)^2$ product code.

3.2. Product Codes as GLDPC Codes

Product codes can also be seen as a generalization of Gallager's LDPC codes. In [30], LDPC codes were originally introduced as codes with a sparse parity-check matrix and single-parity check (SPC) codes as CN constraints. The idea of generalized low density parity check (GLDPC) was brought up by Tanner in [28], where the concept of replacing SPC codes by more powerful linear block codes was established. The iterative decoding of GLDPC codes based on Hamming codes at the CNs was analyzed in [37]. The Tanner graph representation of a product code shows that they are a structured subclass of GLDPC codes. We can see every row and column as a constraint node, or simply a CN. Each bit can be seen as VN and is involved in one row and one column CN. Therefore, VNs have degree 2 and CNs have degree n, assuming the same $\mathcal{C} = (n, k, d)$ component code is used for all rows and columns like in Figure 3.2.
The number of VNs in a GLDPC code is typically defined as N and the number of CNs as M, respectively. For product codes holds that $N = n^2$ and $M = 2n$. However, a product code is only one element of a GLDPC ensemble with design rate R [38]:

$$R = \frac{N - M(n-k)}{N} = 1 - \frac{2(n-k)}{n} = 2\frac{k}{n} - 1. \tag{3.6}$$

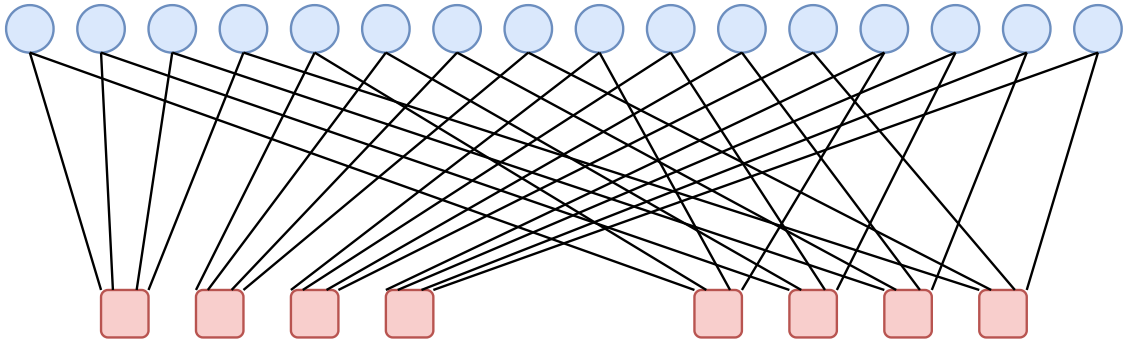


Figure 3.2.: Tanner graph of a product code with component codes of length $n = 4$.

The GLDPC ensemble is defined as the set of codes obtained by selecting all possible edge connections between the N VNs and the M CNs. The design rate of a GLDPC code is derived from its parity check matrix $\boldsymbol{H}$. First of all, we define $\boldsymbol{\Gamma}$ as the adjacency matrix of a Tanner graph. Each VN corresponds to a column and each CN to a row in $\boldsymbol{\Gamma}$. Then we place a one in each position in $\boldsymbol{\Gamma}$ where there is an edge between a VN and CN. For classical LDPC codes, where SPC codes are employed at the CNs, the parity-check matrix $\boldsymbol{H}$ corresponds to $\boldsymbol{\Gamma}$. However, the adjacency matrix of a product code has a special structure, i.e.,

$$\boldsymbol{\Gamma} = \left[\begin{array}{cccc|cccc|cccc|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]. \tag{3.7}$$

To obtain the GLDPC code's parity-check matrix $\boldsymbol{H}$ from $\boldsymbol{\Gamma}$, the ones have to be replaced by the corresponding columns of the parity-check matrix of the component code, which is employed at the CN [23]. Under the assumption that the same component code is used for all CNs, each one in $\boldsymbol{\Gamma}$ is replaced by a column of length $n - k$. We end up with a matrix $\boldsymbol{H}$ with $M(n-k)$ rows and N columns, which leads to the GLDPC rate definition from (3.6). The Tanner graph representation of a product code also makes clear that the edges follow a predefined structure as shown in Figure 3.2. Although this structure allows efficient decoder design, it fixes the length of the shortest cycle in the graph, the so-called *girth*, to 8, regardless its block length [39].

3.3. Hard-Decision Decoding

3.3.1. Iterative Bounded Distance Decoding

In Section 2.5 BDD was introduced as a suboptimal decoding algorithm, which can error patterns with Hamming weight up to the error-correction capability t of the code. We want to formalize this algorithm according to [40, Sec. II.C] and agree on the rule to output $\boldsymbol{r}$, i.e., the decoder input, if there is no codeword in the decoding sphere. We call this case a *failed* BDD attempt. It may also happen that we decide for a wrong codeword $\tilde{\boldsymbol{c}}$, i.e., a *miscorrection* occurs. The BDD behaviors are summarized next, under the assumption that the correct decision is $\boldsymbol{c}$:

$$\hat{\boldsymbol{r}} = \begin{cases} \boldsymbol{c} & \text{if } \mathrm{d_H}(\boldsymbol{c}, \boldsymbol{r}) \leq t \\ \tilde{\boldsymbol{c}} \in \mathcal{C} & \text{if } \mathrm{d_H}(\boldsymbol{c}, \boldsymbol{r}) > t \text{ and } \exists \tilde{\boldsymbol{c}} \text{ such that } \mathrm{d_H}(\tilde{\boldsymbol{c}}, \boldsymbol{r}) \leq t \\ \boldsymbol{r} & \text{otherwise} \end{cases} \tag{3.8}$$

According to (3.4), the minimum distance of a product code $\mathcal{C}$ is given by the product of the minimum distances of the component codes, i.e., $d = d_1 d_2$. The product code construction therefore allows relatively large d resulting in an error correcting capability $t = \left\lfloor \frac{d_1 d_2 - 1}{2} \right\rfloor$ due to (2.39). However, for product codes we also deal with large block-lengths and therefore even suboptimal BDD is infeasible. To decode product codes in a HD decoding manner with an acceptable complexity, iterative bounded distance decoding (iBDD) of the component codes turns out to be a good strategy.

Turning around Elias' procedure of row-column encoding, the original decoding idea of product codes consisted in a single pass of row-column decoding [4]. However, following the same principle of turbo-like codes and LDPC codes, better performance can be achieved by iterating between the component decoders, i.e., by iteratively applying BDD to the row and column codes.

As in Section 3.1 the code bits can be arranged in a matrix form. We denote this codeword by $\boldsymbol{C}$ and obtain the corresponding channel input $\boldsymbol{X}$ by applying the mapping (2.23). The noisy signal at the output of the bi-AWGN channel is denoted by $\boldsymbol{Y}$ and given into the decoder.

In the following iBDD as presented in [40, Alg. 1] is described. At first a hard decision (HD) on the matrix of received channel output symbols $\boldsymbol{Y}$ is taken, resulting in a matrix $\boldsymbol{R} = \mathrm{HD}(\boldsymbol{Y})$ such that $\boldsymbol{R}_{i,j} \in \{0, 1\}$. $\boldsymbol{R}$ has dimension $n_2 \times n_1$ and decoding is carried out on the rows of the array, i.e., BDD decoding of the row component codes $\mathcal{C}_1$ is performed, and subsequently on the columns of the array, i.e., BDD decoding of the column component codes $\mathcal{C}_2$. We refer to the decoding of all rows and all columns as one

iteration. Row-column decoding is then iterated until a maximum number of iterations $I_{\max}$ is reached. Product code encoding and iBDD works a bit different for extended component codes and is addressed in [41, Alg. 1]. We encode the information bits with the standard BCH code with length $2^s - 1$ into a codeword matrix of size $(n_2 - 1) \times (n_1 - 1)$. Afterwards a SPC is performed on each row and column and the resulting parity bit is appended as n-th column and n-th row, respectively. This additional code bit causes a small cost in code rate, but increases the minimum distance of the BCH component by 1. This increased distance can be used to reduce the probability of undetected failure of the component decoder, thereby reducing the number of new errors that are introduced by the component decoder and improving the performance. In the decoding process of product codes with eBCH component codes, we decode each row and column of the $(n_2-1)\times(n_1-1)$ reduced codeword matrix at first. For each decoded row and column, we compute the *estimated* parity bit of the BDD output and check if it matches up with the parity check bit in the n-th row/ column of the recived codeword matrix $\boldsymbol{R}$. Only if the parity bit calculated from the decoded row/ column agrees with the received parity bit in $\boldsymbol{R}$, the decoded row/ column gets updated for the next decoding iteration, otherwise we keep the row/ column from $\boldsymbol{R}$.

3.3.2. Extrinsic Message Passing Decoding

Clearly iBDD violates the concept of only exchanging *extrinsic* information [42]. When treating the product code as GLDPC code, the CN update rule for a certain outgoing edge should be independent of what came into the CN over this edge. This is clearly not the case for the described iBDD algorithm, because a row or column is corrected on the basis of the whole codeword, i.e., all incoming edges of the CN. Therefore in [38, Sec. VIII] iBDD is also called intrinsic message passing (IMP) and an extrinsic version of IMP was derived. The algorithm is called extrinsic message passing (EMP) and achieves slightly better performance compared to IMP.
A CN update for EMP works as follows. Let $\nu_{i,j}^{(I)} \in \{0,1\}$ be the message passed by the EMP algorithm from the i-th VN to the j-th CN and let $\boldsymbol{\nu}_j^{(I)} \triangleq \left(\nu_{\sigma_j(1),j}^{(I)}, \ldots, \nu_{\sigma_j(n),j}^{(I)}\right)$ be the collection of all input messages at the j-th CN in the I-th decoding iteration. To compute the CN output message $\mu_{i,j}^{(I)} \triangleq \left(\mu_{i,j,0}^{(I)}, \mu_{i,j,1}^{(I)}\right)$ from CN j to VN i, BDD is performed twice. The inputs for the BDD decoder are defined as

$$\boldsymbol{\nu}_{j,k,0}^{(I)} \triangleq \left(\nu_{\sigma_j(1),j}^{(I)}, \ldots, \nu_{\sigma_j(k-1),j}^{(I)}, 0, \nu_{\sigma_j(k+1),j}^{(I)}, \ldots, \nu_{\sigma_j(n),j}^{(I)}\right) \tag{3.9}$$

and

$$\boldsymbol{\nu}_{j,k,1}^{(I)} \triangleq \left(\nu_{\sigma_j(1),j}^{(I)}, \ldots, \nu_{\sigma_j(k-1),j}^{(I)}, 1, \nu_{\sigma_j(k+1),j}^{(I)}, \ldots, \nu_{\sigma_j(n),j}^{(I)}\right). \tag{3.10}$$

The BDD decoder therefore ignores the a priori information on the incoming edge from VN i and tests both possible codeword combinations. Since the BDD decoder may decide for 0,1 or fail, if no codeword is in the bounded distance sphere around the decoder input, the CN output messages $\mu_{i,j}^{(I)} = \left(\mu_{i,j,0}^{(I)}, \mu_{i,j,1}^{(I)}\right)$ will be in the set

$$\{(0,0),(1,1),(0,1),(0,\text{fail}),(\text{fail},1),(\text{fail},\text{fail})\}. \tag{3.11}$$

At the VN i the message passing rule to the other CN in the neighborhood of i ($\mathcal{N}(i) = \{j, j'\}$ for product codes) is computed as

$$\nu_{i,j'}^{(I+1)} \triangleq \begin{cases} 0 & \text{if } \mu_{i,j}^{(I)} = (0,0) \\ 1 & \text{if } \mu_{i,j}^{(I)} = (1,1) \\ r_i & \text{otherwise .} \end{cases} \tag{3.12}$$

Note that this extrinsic version of iBDD entails a significant increase in complexity. If we assume that the length of row and column code are equal ($n_1 = n_2 = n$), the number of BDD attempts per iteration grows from $n + n = 2n$ for iBDD to $n \cdot 2n + n \cdot 2n = 4n^2$ for EMP. This comes from the modification that a decision was built row-/ column wise

for iBDD, whereas for EMP a bit-wise decision for each row and column is made based on two possible combinations (zero or one) of the bit position in the corresponding row/ column.

As a benchmark or reference for iBDD, the *ideal* iBDD approach can be used. Ideal iBDD is a genie-aided version of iBDD avoiding miscorrections, i.e., the BDD decoder either finds the correct codeword of a row/ column or it outputs $\boldsymbol{r}$. If the BDD algorithm maps onto a wrong codeword, the genie would intervene and output $\boldsymbol{r}$ instead.

There has been considerable research on different variations of iBDD in the past years to close the gap between conventional iBDD and ideal iBDD. An approach called anchor decoding (AD) exploits conflicts between component codes in order to assess reliabilities on the conflicting bit positions [40].

Another approach with exploitation of the channel soft information output is called iterative bounded distance decoding with scaled reliability (iBDD-SR) [43]. In iBDD-SR hard messages from a ternary alphabet are exchanged between the row- and column decoders. The messages are formed based on the sum of a weighted BDD output and the channel log-likelihood ratio (LLR). A refined version of iBDD-SR which builds a more accurate combination of the BDD output and the channel LLR, is called iterative bounded distance decoding with combined reliability (iBDD-CR) [44]. iBDD-CR was quite recently proposed and outperforms ideal iBDD while keeping the same internal decoder data flow as that of iBDD.

The concept of iBDD can be further improved if generalized minimum distance decoding (GMDD) [45] instead of BDD is applied. The idea of GMDD is to form a set of trial vectors based on the channel LLR of a row or column, perform error-erasure decoding to each vector and form a list of candidate codewords. From this list the codeword which minimizes the generalized distance is chosen. The resulting algorithms are called binary message passing decoding based on GMDD (BMP-GMDD) [46] and iterative generalized minimum distance decoding with scaled reliability (iGMDD-SR) [47]. They show a significant performance gain compared to the less complex iterative algorithms with BDD core decoders.

3.4. Soft-Decision Decoding

With the introduction of turbo codes in 1993 [29], it was obvious that the iterative decoding structure of product codes is suitable for an adaption to the turbo principle of exchanging extrinsic information between two soft-input soft-output (SISO) decoders. The optimal BCJR algorithm [13], which might be used to generate extrinsic information, turned out to be an impossible solution for practical high-throughput systems, as the number of states in the code trellis grows exponentially with the number of redundancy bits. Based on the much less complex Chase decoder [48], Ramesh Pyndiah finally came up with a novel decoder in 1998 [5]. The so-called Chase-Pyndiah (Ch-Py) algorithm is in the main scope of the remainder of this thesis and introduced in this section, which is organized as follows. The TPD principle and SISO decoding are introduced in 3.4.1 at first. In 3.4.2 the procedure of Chase decoding is explained and illustrated based on a numerical example. Finally both concepts, TPD and the Chase algorithm, are combined in 3.4.3 to the Ch-Py algorithm, for which several simulation results are provided.

3.4.1. The Turbo Product Decoding Principle

In [5] turbo product decoding (TPD) was introduced as an iterative exchange of reliability on the code bits between a row and column SISO decoder. Assume a codeword $\boldsymbol{c}$ is mapped to a channel input $\boldsymbol{x}$ according to (2.23) and transmitted over a bi-AWGN channel, where the channel output $\boldsymbol{y}$ is detected. In [35, Cha. 4] a SISO module is described by a decoder with 2 inputs and 2 outputs. The first input is the channel output y_i, which is transformed into the LLR domain by

$$\mathrm{L}_i^{\mathrm{CH}} = \frac{2}{\sigma^2} y_i. \tag{3.13}$$

The second input is the a priori LLR $\mathrm{L}_i^{\mathrm{A}}$. Then the extrinsic LLR can be computed as

$$\mathrm{L}_i^{\mathrm{E}} = \mathrm{L}_i^{\mathrm{APP}} - (\mathrm{L}_i^{\mathrm{CH}} + \mathrm{L}_i^{\mathrm{A}}), \tag{3.14}$$

where $\mathrm{L}_i^{\mathrm{APP}}$ is the APP LLR on code bit i, defined as

$$\mathrm{L}^{\mathrm{APP}}(c_i) = \ln\left(\frac{p(x_i = +1|\boldsymbol{y})}{p(x_i = -1|\boldsymbol{y})}\right). \tag{3.15}$$

The a priori information for the column decoder corresponds to the extrinsic information generated by the row decoder, i.e., $\mathrm{L}^{\mathrm{A,col}} = \mathrm{L}^{\mathrm{E,row}}$ and vice versa, i.e., $\mathrm{L}^{\mathrm{A,row}} = \mathrm{L}^{\mathrm{E,col}}$. This procedure is visualized in Figure 3.3 and may be carried on for a desired number of

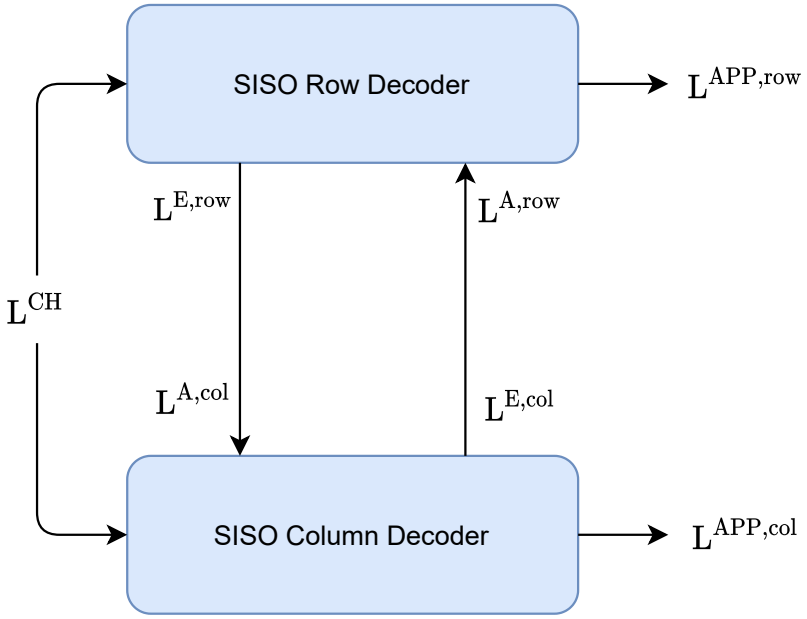


Figure 3.3.: The Turbo Product Decoding Principle.

iterations. The APP LLR in the last iteration is used for the final binary decision.

3.4.2. The Chase Algorithm

In 1972 David Chase has presented a SD decoder in [48], which offers an appealing alternative to using the whole codebook $\mathcal{C}$. Pyndiah used the framework of this decoder in [5] to lower the complexity of the SISO modules knowing that the Chase algorithm just yields an **approximation** of the true a-posteriori probability. We want to emphasize that also the computed reliability on a code bit is only **approximated** extrinsic information. Nevertheless, due to reasons of correlation, we will stick to the notation of L^E in the following.

The Chase algorithm describes a way to build a list of candidate codewords $\mathcal{L}$ in order to approximate the true APP as good as possible. Assume a vector of LLRs is given as input $\mathbf{L}^{IN}$ into the Chase decoder. Then the steps to compute the reliability for each bit position are explained below and also visualized based on a numerical example in Figure 3.4.

1.) Find the p least reliable bit positions (LRBP) in the decoder input LLR vector $\mathbf{L}^{\mathrm{IN}}$.
2.) Create a test list of size 2^p by including $\boldsymbol{r}$ =HD($\mathbf{L}^{\mathrm{IN}}$) and $\boldsymbol{r}$ flipped on the $1 \dots p$ LRBPs.
3.) Run BDD on every testword in the test list. Include the output of the BDD decoder to the candidate list $\mathcal{L}$ only if the BDD attempt was successful and if the BDD output is not already contained in $\mathcal{L}$. If $\mathcal{L} = \emptyset$ after decoding all testwords, set $\mathcal{L} = \boldsymbol{r}$.
4.) Find the codeword $\boldsymbol{d}$ in $\mathcal{L}$ with minimum squared Euclidean distance to $\mathbf{L}^{\mathrm{IN}}$.

$$\boldsymbol{d} = \min_{\tilde{\boldsymbol{x}} \in \mathcal{L}} ||\mathbf{L}^{\mathrm{IN}} - \tilde{\boldsymbol{x}}||^2 \tag{3.16}$$

5.) Compute the reliability on each code bit $i = 1 \dots n$. For this, a competing codeword $\bar{\boldsymbol{c}}$ to $\boldsymbol{d}$ needs to be found for each bit position.
If $r_i = 0$, we received a $+1$ according to (2.23) in bit position i and consequently the competing list $\mathcal{L}^{-,i}$ is a sublist of $\mathcal{L}$ only containing -1, i.e., the opposite sign in the i-th bit position.
The competing codeword is the entry of the competing list $\mathcal{L}^{-,i}$ with minimum squared euclidean distance to $\mathbf{L}^{\mathrm{IN}}$.
If such a competing codeword $\bar{\boldsymbol{c}}$ is found, then $\mathbf{L}^{\mathrm{E}}$ can be calculated as

$$\mathrm{L}_i^{\mathrm{E}} = \frac{1}{2} \left(\sum_{j \neq i}^{n} \mathrm{L}_j^{\mathrm{IN}} \left(d_j - \bar{c}_j \right) \right) d_i. \tag{3.17}$$

If the competing list is empty for a certain bit position i, then the decoder simply outputs a weighted version of the decision codeword $\boldsymbol{d}$ at position i, i.e.,

$$\mathrm{L}_i^{\mathrm{E}} = \beta d_i. \tag{3.18}$$

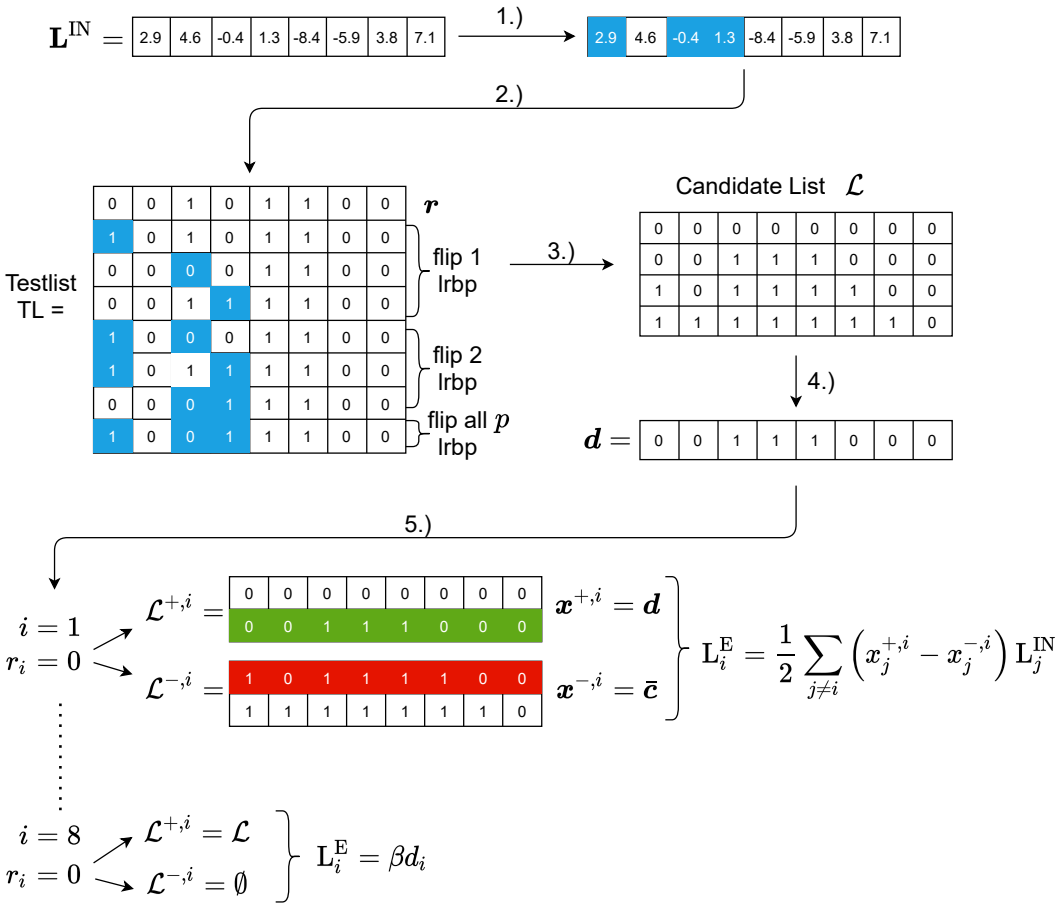


Figure 3.4.: Numerical example of a Chase Decoder.

In literature about Ch-Py [5, 49, 50], there are different formulas for the computation of $\mathbf{L}^{\mathrm{E}}$. To avoid confusion concerning various expressions, the equivalence of the following three terms will be shown.

$$\begin{aligned} \mathrm{L}_i^{\mathrm{E}} &\overset{(1)}{=} \frac{1}{2}\left(\sum_{j\neq i}^{n} \mathrm{L}_j^{\mathrm{IN}}\left(d_j - \bar{c}_j\right)\right) d_i \\ &\overset{(2)}{=} \frac{1}{2}\left(\sum_{j\neq i}^{n} \mathrm{L}_j^{\mathrm{IN}}\left(x_j^{+,i} - x_j^{-,i}\right)\right) \\ &\overset{(3)}{=} \left(\frac{||\mathbf{L}^{\mathrm{IN}} - \bar{\boldsymbol{c}}||^2 - ||\mathbf{L}^{\mathrm{IN}} - \boldsymbol{d}||^2}{4}\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \end{aligned} \tag{3.19}$$

Term 1 and 2 are equivalent, because $\boldsymbol{d}$ is always either $\boldsymbol{x}^{+,i}$ or $\boldsymbol{x}^{-,i}$, depending on the sign of d_i, because it is the closest codeword to $\mathbf{L}^{\mathrm{IN}}$ in the candidate list.
If $d_i = +1$, then $\boldsymbol{x}^{+,i} = \boldsymbol{d}$ and $\boldsymbol{x}^{-,i} = \bar{\boldsymbol{c}}$. Thus, $(\boldsymbol{x}^{+,i} - \boldsymbol{x}^{-,i}) = (\boldsymbol{d} - \bar{\boldsymbol{c}})$.
If $d_i = -1$, then $\boldsymbol{x}^{+,i} = \bar{\boldsymbol{c}}$ and $\boldsymbol{x}^{-,i} = \boldsymbol{d}$. Thus, $(\boldsymbol{x}^{+,i} - \boldsymbol{x}^{-,i}) = (\bar{\boldsymbol{c}} - \boldsymbol{d}) = -(\boldsymbol{d} - \bar{\boldsymbol{c}})$.
To ensure equality d_i can be used as a correction factor, i.e., $-(\boldsymbol{d} - \bar{\boldsymbol{c}})d_i = (\boldsymbol{d} - \bar{\boldsymbol{c}})$.

The equivalence of term 3, as given in [5, Eq. 18] is not obvious at first glance, but becomes clear after expansion of the squared Euclidean distances.

$$\begin{aligned} \mathrm{L}_i^{\mathrm{E}} &= \left(\frac{||\mathbf{L}^{\mathrm{IN}} - \bar{\boldsymbol{c}}||^2 - ||\mathbf{L}^{\mathrm{IN}} - \boldsymbol{d}||^2}{4}\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \\ &= \left(\frac{\sum_{j=1}^{n}\left(\mathrm{L}_j^{\mathrm{IN}} - \bar{c}_j\right)^2 - \left(\mathrm{L}_j^{\mathrm{IN}} - d_j\right)^2}{4}\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \\ &= \left(\frac{\sum_{j=1}^{n}\left(\mathrm{L}_j^{\mathrm{IN}}\right)^2 - 2\mathrm{L}_j^{\mathrm{IN}}\bar{c}_j + \bar{c}_j^2 - \left(\left(\mathrm{L}_j^{\mathrm{IN}}\right)^2 - 2\mathrm{L}_j^{\mathrm{IN}}d_j + d_j^2\right)}{4}\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \\ &= \left(\frac{\sum_{j=1}^{n}\left(\mathrm{L}_j^{\mathrm{IN}}\right)^2 - 2\mathrm{L}_j^{\mathrm{IN}}\bar{c}_j + 1 - \left(\mathrm{L}_j^{\mathrm{IN}}\right)^2 + 2\mathrm{L}_j^{\mathrm{IN}}d_j - 1}{4}\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \\ &= \left(\frac{\sum_{j=1}^{n} 2\mathrm{L}_j^{\mathrm{IN}}\left(d_j - \bar{c}_j\right)}{4}\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \\ &= \frac{1}{2}\left(\sum_{j=1}^{n} \mathrm{L}_j^{\mathrm{IN}}\left(d_j - \bar{c}_j\right)\right) d_i - \mathrm{L}_i^{\mathrm{IN}} \\ &= \frac{1}{2}\left(\sum_{j\neq i}^{n} \mathrm{L}_j^{\mathrm{IN}}\left(d_j - \bar{c}_j\right)\right) d_i \end{aligned} \tag{3.20}$$

3.4.3. Chase-Pyndiah Decoding

The Chase-Pyndiah Algorithm

Pyndiah combined in [5] the concepts of the two previous subsections by employing a Chase decoder as SISO component decoder in the iterative fashion described in Figure 3.3. The final binary decision on the codebits is usually not taken after only one iteration, especially because there is no a-priori information available, yet. We denote by a decoding iteration, when all rows and columns of the codeword matrix are decoded. According to the turbo principle, in each iteration the input LLR $\mathbf{L}^{\mathrm{IN}}$ for a row/ column SISO decoder is simply given by the sum of the channel LLR $\mathbf{L}^{\mathrm{CH}}$ and the a-priori information $\mathbf{L}^{\mathrm{A}}$. However, due to the suboptimal nature of the Chase algorithm, the computed reliability, which is just an approximation of the true extrinsic information, tends to highly overestimate the decision for a certain bit position. Pyndiah came up with a method of scaling the extrinsic information before passing it to the other SISO decoder to cope with this issue. He distinguished between two different cases and even provided a numerical choice for the corresponding scaling factors.

- α = scaling factor for the extrinsic information computed according to (3.17).

$$\alpha = [0.0, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0, \ldots, 1.0] \tag{3.21}$$

- β = reliability factor on the decision codeword bit d_i, if the competing list is empty (3.18).

$$\beta = [0.2, 0.4, 0.6, 0.8, 1.0, 1.0, 1.0, \ldots, 1.0] \tag{3.22}$$

Note that the entries of α, β correspond to half-iterations, i.e., a single row/ column decoding stage and thus $\alpha(1)$ (first half-iteration = no a-priori information) is set to 0. Now let us denote the extrinsic information values from (3.17) by $\mathrm{L}^{\mathrm{E},\,\alpha}$ and from (3.18) by $\mathrm{L}^{\mathrm{E},\beta}$, respectively. Pyndiah stated in [5, Sec. VI] that not scaling $\mathrm{L}^{\mathrm{E},\alpha}$ by α, but the combination of scaling and normalizing the mean absolute value to one, is crucial for the FEC performance. To reduce the dependency of parameter β, the mean absolute value of all $\mathbf{L}^{\mathrm{E},\alpha}$ values is normalized to one in each iteration. This means that before passing $\mathrm{L}^{\mathrm{E},\alpha}$ as $\mathrm{L}^{\mathrm{A},\alpha}$ to the next SISO decoder, we must normalize according to

$$\mathbf{L}^{\mathrm{E},\alpha,\mathrm{norm}} = \frac{\mathbf{L}^{\mathrm{E},\alpha}}{\mathrm{mean}(|\mathbf{L}^{\mathrm{E},\alpha}|)} \tag{3.23}$$

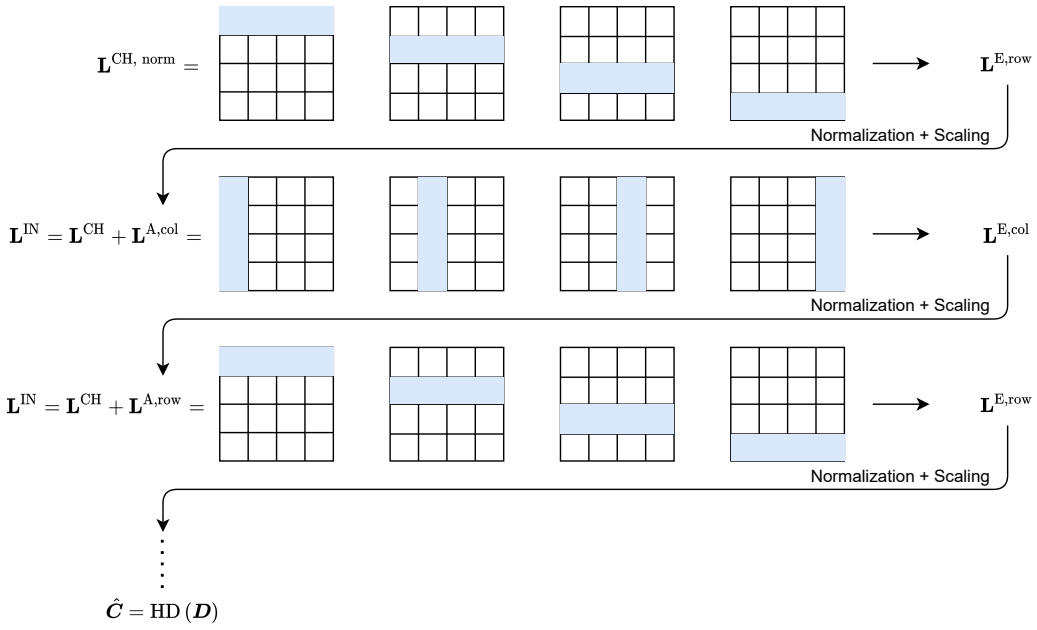


Figure 3.5.: Iterative Fashion of standard Chase-Pyndiah Turbo Product Decoding.

and then scale with α to obtain

$$\mathbf{L}^{\mathrm{A},\alpha} = \alpha \cdot \mathbf{L}^{\mathrm{E},\alpha,\mathrm{norm}}. \tag{3.24}$$

The decoder therefore has to be fed with a channel LLR $\mathbf{L}^{\mathrm{CH}}$ normalized in the same manner initially. Although this normalization leads to a loss of information about the magnitudes of the reliability values, it allowed Pyndiah to provide scaling factors, which are independent of the used component code and applicable over the whole SNR range without facing numerical issues. The complete procedure is illustrated in Figure 3.5 for three half-iterations. According to [5] the final binary decision in the last half-iteration is taken based on the most likely codeword $\boldsymbol{d}$ ($\boldsymbol{D}$ in matrix form) in the candidate list $\mathcal{L}$ of the Chase decoder.

Choice of Parameters

There are three parameters, where we can control the Ch-Py decoding performance-complexity trade-off:

- the number of LRBP p on which the Chase decoder test list is built on.
- the choice of the component code decoder, which decodes all test words in the test list and forms the candidate list $\mathcal{L}$ (e.g. BDD or ML).
- maximum number of decoding iterations $I_{\max}$ between row and column SISO decoders.

In the following the impact of these parameters on the FEC performance will be evaluated based on simulations over the bi-AWGN channel.
In Figure 3.6 the gain of using a higher number of p LRBP in the Chase decoder is shown. A product code with (64,51,6) eBCH component code with error correcting capability $t = 2$ is simulated for a maximum of 8 decoding iterations. Pyndiah's choice of $p = 4$ turns out to be a good trade-off, since there is a significant performance gain compared to $p = 3$, but the smaller gain from $p = 4$ to $p = 5$ is not made up by the huge increase in complexity. Note that for each row/ column, the Chase algorithm needs 2^p BDD attempts to decode all the test words to form a candidate list $\mathcal{L}$.
In Figure 3.7 the choice of the algebraic hard decoder is discussed. The choice was left open by Pyndiah, but for practical applications a ML decoder becomes infeasible already for component codes such as the simulated (32,21,6) eBCH code. As shown in Figure 2.5, BCH codes perform very close to ML, when decoded via BDD. As a result of that, the gain of using ML to decode the test list is very small compared to BDD. However, for different specific component codes there exist more advantageous version of the Chase decoder as proposed in [51, 52].
In Figure 3.8, the impact of allowing a higher number of maximum decoding iterations $I_{\max}$ is depicted. Based on a product code with (32,26,4) extended Hamming codes as component codes, the significant gain from decoding iteration 1 to 4 can be observed. After iteration 4, the FEC performance is not much enhanced any more, because the error statistics are dominated by stall patterns, which cannot be resolved by more decoding iterations between the SISO decoders. Employing a larger number of $I_{\max}$ also requires an efficient stopping criterion in order to prevent unnecessary computations in the decoder [53]. On the right hand side of Figure 3.8, where the FER is plotted, the truncated UB of the $(32, 26, 4)^2$ product code indicates that Ch-Py provides close-to-optimal performance in the error floor region for this specific choice of component codes.

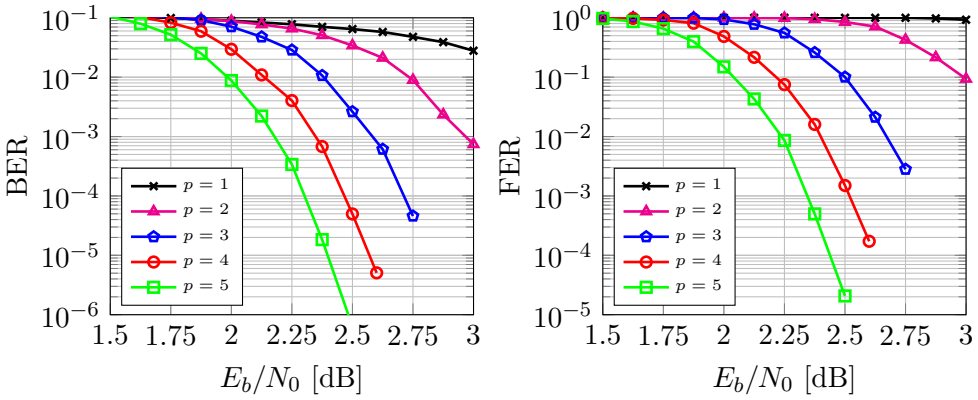


Figure 3.6.: Gain of using a higher number p of LRBP in the Chase-Pyndiah algorithm evaluated with a $(64, 51, 6)^2$ product code with $I_{\max} = 8$.

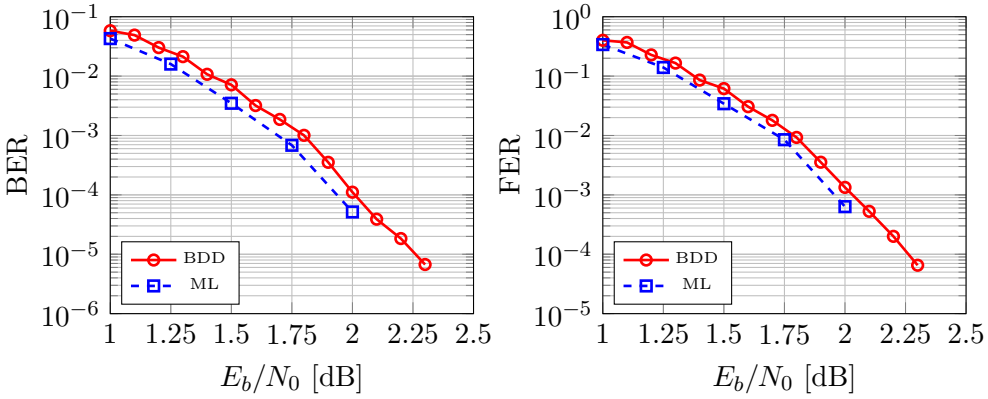


Figure 3.7.: Comparison between a BDD and ML component decoder in the Chase-Pyndiah algorithm evaluated with a $(32, 21, 6)^2$ product code with $I_{\max} = 8$ and $p = 4$.

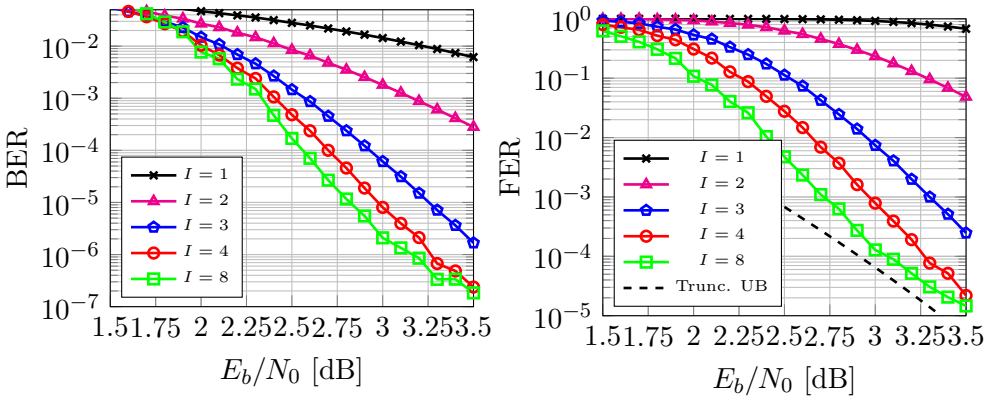


Figure 3.8.: Gain of using a higher number of iterations $I_{\max}$ in the Chase-Pyndiah algorithm evaluated with a $(32, 26, 4)^2$ product code with $p = 4$.

Standard Chase Pyndiah Decoding

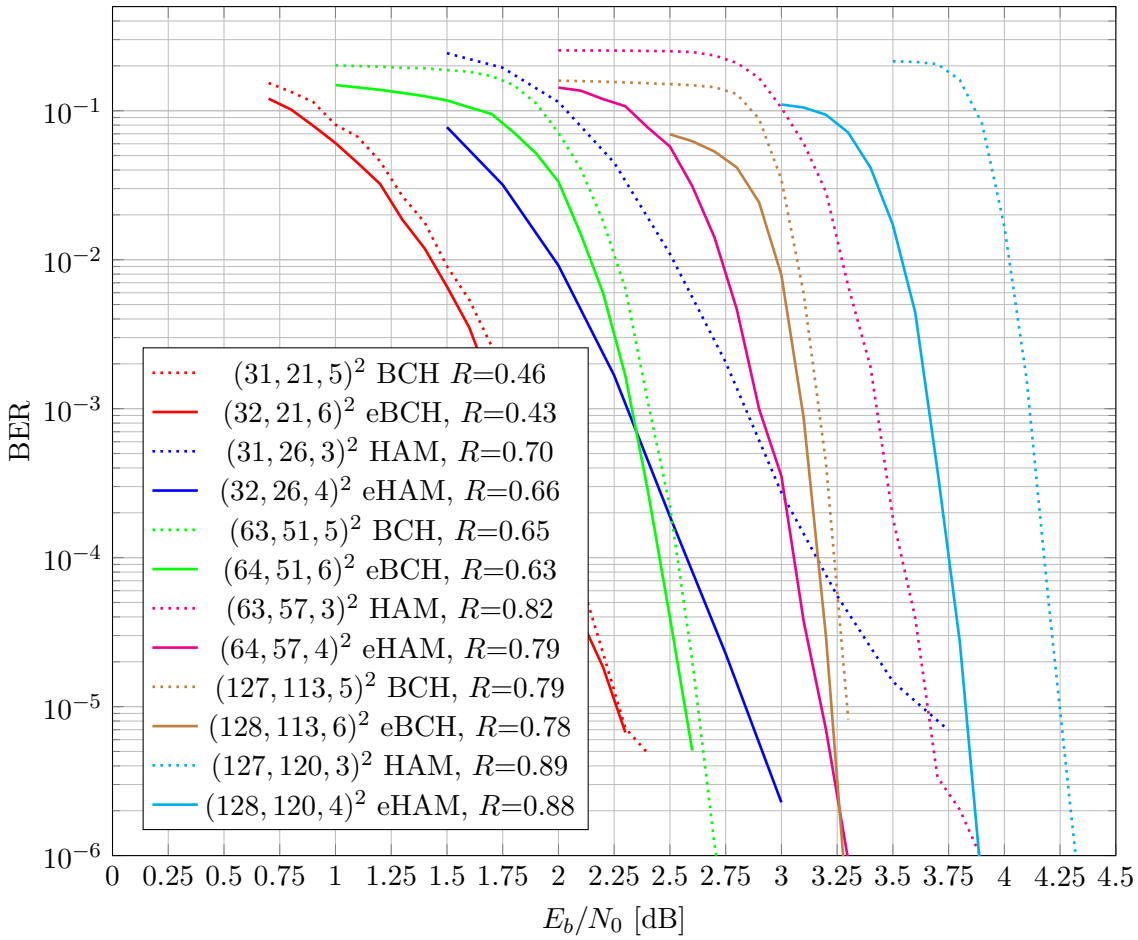


Figure 3.9.: Prodcut Codes with different (e)BCH component codes decoded via standard Chase-Pyndiah and $I_{\max} = 8$.

If we refer to *standard* Ch-Py decoding in the remainder of this thesis, the setup from [5] is used, which is charcterized by:

ALGORITHM 1

Standard Chase-Pyndiah Decoding

- Chase decoder with $p = 4$ LRBP and BDD decoding of test list.
- Pyndiah's scaling factors (α from (3.21), β from (3.22)).
- Normalization of L^{CH} and $\mathrm{L}^{\mathrm{E},\alpha}$ according to (3.23).
- Final binary decision based on most liekly codeword $\boldsymbol{D}$ of Chase decoder.

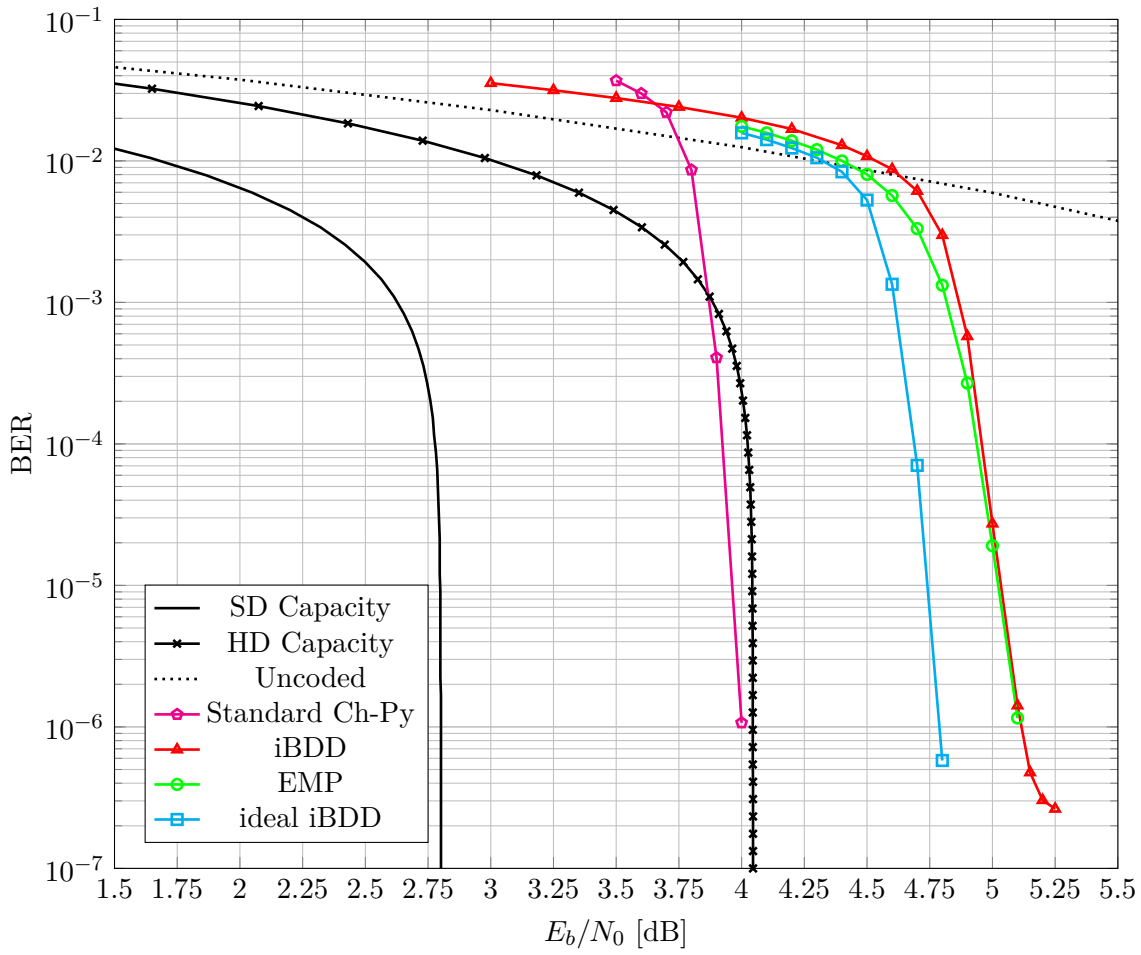


Figure 3.10.: Prodcut Code with (256,239,6) eBCH component code decoded via different decoding algorithms at $I_{\max} = 10$ and their gap to capacity.

In Figure 3.9 we have reproduced the BER results from [5, 50] of standard Ch-Py to ensure a correct implementation of the algorithm. The decoder was applied to product codes with various component codes. Clearly, the difference in code rate has to be taken into account when comparing the curves. We observe that the gain of using product codes with extended Hamming codes instead of normal Hamming codes is massive compared to the small loss in code rate. This fact only seems to hold for Hamming codes ($t = 1$), because the difference between BCH and eBCH component codes with $t > 1$ is relatively small. Though, only product codes with eBCH and extended Hamming component codes will be analyzed in the remainder of this thesis.
To show the performance of standard Ch-Py in contrast to other decoding algorithms for product codes presented in this chapter, the BER performance for a product code with

(256,239) eBCH codes with error correcting capability $t = 2$ is shown in Figure 3.10. Except from the EMP performance curve, the results have also already been reported in [47]. Next to the simulated codes, the SD and HD bi-AWGN Shannon limit for the corresponding code rate $R = 0.8716$ derived in Section 2.4.2 is plotted. We can observe that standard Ch-Py performs about 1.25 dB away from SD capacity in the waterfall region. For the HD decoding algorithm iBDD, we can see the genie-aided ideal iBDD (with no BDD miscorrections) as a benchmark. It becomes clear that much more complex EMP is not able to close the gap between iBDD and ideal iBDD. Thanks to the large codelength, the gap between iBDD and the HD capacity is only approximately 1.1 dB.

4. Parameter Optimization for Chase-Pyndiah Decoding

In this chapter, we want to evaluate Pyndiah's choice of parameters for standard Ch-Py decoding from Algorithm 1 and optimize them. The parameters were originally found by trial and error and designed for an application over a large range of codes at all SNRs [5]. We want to find out if and how much performance gain in terms of BER may be achieved by scaling dynamically. This means instead of using predefined coefficients, they are computed instantaneously in the decoding process without incurring a large increase in complexity. The first optimization method in Section 4.1 can be directly applied to the standard Ch-Py algorithm and is based on a heuristic approach to scale the extrinsic information with the fraction of valid codewords in the product code matrix. The second method in Section 4.2 follows a different strategy which has the goal to make the use of scaling factors unnecessary. Instead of applying a predefined attenuation α to the decoder output, postprocessing with a function dependent on the current convergence state, is proposed.

4.1. Scaling Extrinsic Information with the Fraction of Valid Codewords

In this section we present the method of scaling L^{E} with the fraction of valid codewords. We propose this metric to be used as an indicator of how close or how far the decoded codeword matrix is away from a valid codeword of the product code. Computing this fraction in each half-iteration gives us the chance to react on possible divergence behavior over the decoding iterations by stronger attenuation of the extrinsic output. Although there is no analytical evidence of optimality, the method yields a remarkable performance gain without an increase in computational complexity.

An efficient stopping criterion to stop decoding before $I_{\max}$ is reached, is to check if all rows and columns are valid codewords of the employed component code [53]. If employing this criterion, the syndromes of all rows and columns are computed anyway in each half-iteration and the calculation of the fraction of valid codewords does not incur major

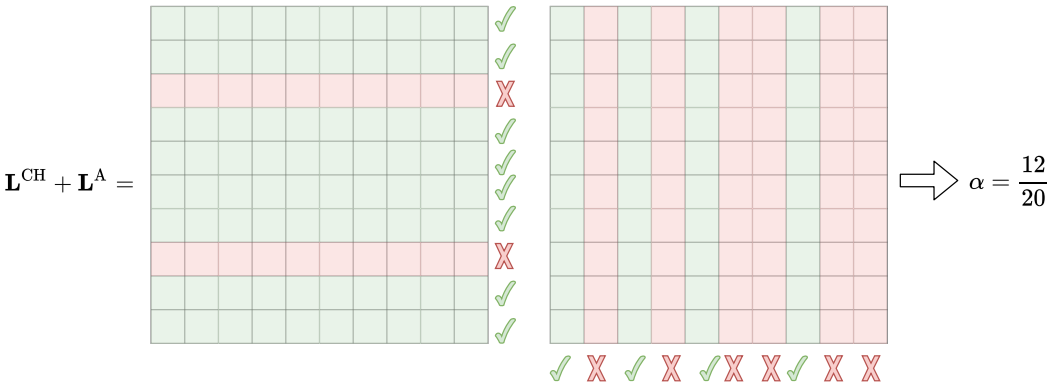


Figure 4.1.: Determining α as the fraction of valid codewords.

additional computational costs. To the best of the author's knowledge such a strategy has not been considered for Ch-Py in literature before. In [54, Sec. V] the authors proposed to use the fraction of unsatisfied checks of an LDPC code as the extrinsic error probability of a CN. Therefore, we do not want to claim that the underlying idea is completely new.

The proposed method of scaling may be integrated in a straight-forward manner into the standard Ch-Py Algorithm 1. Assume the transmission of a codeword matrix $\boldsymbol{C}$ over a bi-AWGN channel. We receive the channel output matrix $\boldsymbol{Y}$ and compute the number of rows and columns of $\boldsymbol{R}$ =HD($\boldsymbol{Y}$), which are valid codewords of the component code, i.e., they satisfy (2.35) and hence the syndrome $\boldsymbol{s} = 0$. This number divided by the number of all component code codewords ($n_{\mathrm{row}} + n_{\mathrm{col}}$) gives the fraction of valid codewords used for scaling the Chase decoder output in the first half-iteration. For the second half iteration, the fraction of valid codewords is built on HD($\mathbf{L}^{\mathrm{IN}}$), where $\mathbf{L}^{\mathrm{IN}} = \mathbf{L}^{\mathrm{CH}} + \mathbf{L}^{\mathrm{A}}$. Thus α is obtained by

$$\alpha = \frac{|\text{rows of } \mathbf{L}^{\mathrm{IN}} \text{ with } \boldsymbol{s} = 0| + |\text{columns of } \mathbf{L}^{\mathrm{IN}} \text{ with } \boldsymbol{s} = 0|}{n_{\mathrm{row}} + n_{\mathrm{col}}}. \tag{4.1}$$

In Figure 4.1 a numerical example is shown, where all but 2 rows satisfy the valid codeword condition, but only 4 out of 10 columns do. Therefore 12 out of 20 component code codewords are valid, which leads to $\alpha = 12/20$.

If all rows and columns are valid, $\alpha = 1$ and decoding stops. The fraction of valid codewords may be seen as a measure of convergence, i.e., we trust the decoder output more, if there are more valid codewords in the product code matrix. Especially in the first iterations, where not many rows or columns have a syndrome equal to zero, we should attenuate the extrinsic decoder output more, similar to (3.21).

Combining (4.1) and Algorithm 1, we can formalize a new modified Ch-Py algorithm as follows.

ALGORITHM 2

Chase-Pyndiah Decoding with Extrinsic Information Scaling by the Fraction of Valid Codewords

- Chase decoder with $p = 4$ LRBPs and BDD decoding of test list.
- Dynamically adaptive α resembling the fraction of valid codewords (4.1).
- Pyndiah's scaling factor β from (3.22).
- Normalization of L^{CH} and $\mathrm{L}^{\mathrm{E},\alpha}$ according to (3.23).
- Final binary decision based on most liekly codeword $\boldsymbol{D}$ of Chase decoder.
- Early stopping when all syndromes equal to zero.

The results in Figure 4.2 are evaluated at $I_{\max} = 8$ and the Chase decoder is run with $p = 4$ LRBPs. For the different product codes from Figure 3.9 the results of the proposed method of computing α as the fraction of valid codewords decoding iterations (Alg. 2) are compared to standard Ch-Py (Alg. 1). All the product codes with eBCH component codes with $t = 2$ show a gain when decoded via Algorithm 2 compared to standard Ch-Py. Especially the $(64, 51, 6)^2$ product code shows a remarkable improvement.
However, for product codes with extended Hamming component codes with $t = 1$, this gain is only visible for lower SNR. For larger blocklengths like the $(128, 120, 4)^2$ product code, Algorithm 2 even performs a bit worse than standard Ch-Py. Since the syndromes are computed anyways if early stopping according to [53] is applied, Algorithm 2 incurs only a very small increase in complexity compared to Algorithm 1 in terms of decoder design.

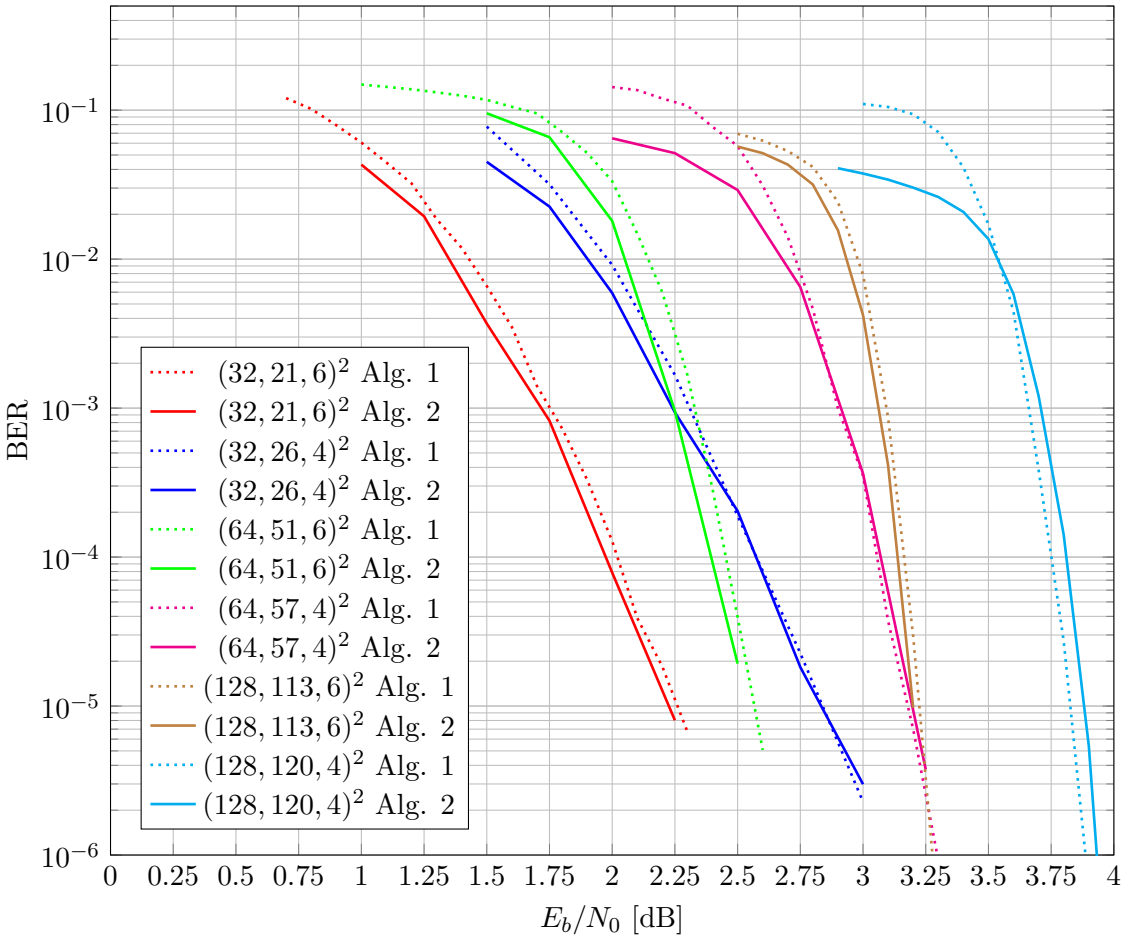


Figure 4.2.: Results of modified Chase-Pyndiah with scaling extrinsic information by the fraction of valid codewords (Alg. 2) compared to standard Chase-Pyndiah (Alg. 1) with $I_{\max} = 8$.

4.2. Extrinsic Information Postprocessing

In this section, a different approach compared to scaling with α and β as in Algorithms 1 and 2 is investigated. We restrict ourselves to the decoding analysis of the component codes at first. Our aim is to derive a postprocessing function, which maps the soft output generated by a Chase decoder closer to the optimal value in order to make the use of a scaling factor α unnecessary.
For LDPC codes, such an approach is already well known. In this case the optimal output from the sum-product algorithm can be compared with suboptimal min-sum decoding. The relation between the extrinsic output of the two decoders was analyzed in [9, Sec. 5.5.2] and the idea to derive a postprocessing function for the min-sum decoder based on the connection to the true MAP output was proposed in [55]. However, due to the more complex structure of the Chase decoder, an analytical relation between the Chase and MAP output cannot be derived. In this section such a relation is established based on the generation of Chase and MAP extrinsic information sample pairs. With the help of these sample pairs, a postprocessing function is derived and integrated in the iterative Chase-Pyndiah decoding process.

4.2.1. Derivation of a Postprocessing Function

To get a good approximation of the relation between L^{E} values of Chase and MAP decoding, a large number of samples has to be generated at first. The strategy is to simulate for a fixed SNR level and a fixed number p of LRBPs, extrinsic output values from a Chase decoder ($\mathrm{L}^{\mathrm{E,Chase}}$) and from a MAP decoder ($\mathrm{L}^{\mathrm{E,MAP}}$), based on the same channel LLR L^{CH} samples and the same linear block code. In our case we only analyze stand-alone BCH codes and thus the channel noise variance σ^2_{CH} must be computed according to (2.22) with respect to the component BCH code rate R_{BCH}.
For the generation of the MAP samples, we can use for example a Hartmann-Rudolph (HR) decoder [14]. The idea of HR decoding is to calculate the APP based on all codewords of the dual codebook. In coding systems where the dual code has fewer codewords than the original code, this results in a reduction of the decoding complexity. When using the original formula in [14], we have to be aware that Hartmann and Rudolph were only interested in finding a symbol-by-symbol MAP decoder with hard outputs and they did not investigate either soft-output or a priori information. Both are crucial for iterative decoding. The true APP on the k-th bit position of the channel output of a codeword encoded with an (N, K) binary linear block code and decoded with an HR decoder has

been derived in [56, Sec. III,E] and is given by

$$\mathrm{L}_k^{\mathrm{APP}} = \mathrm{L}_k^{\mathrm{CH}} + \underbrace{\log \frac{1 + \sum_{i=2}^{2^{N-K}} \prod_{j=1, j\neq k}^{N} \left(\tanh\left(\mathrm{L}_j^{\mathrm{CH}}/2\right)\right)^{\left(1-x'_{ij}\right)/2}}{1 - \sum_{i=2}^{2^{N-K}} \left(-x'_{ik}\right) \prod_{j=1, j\neq k}^{N} \left(\tanh\left(\mathrm{L}_j^{\mathrm{CH}}/2\right)\right)^{\left(1-x'_{ij}\right)/2}}}_{\mathrm{L}_k^{\mathrm{E}}}. \tag{4.2}$$

The computation is complex and not feasible for codes with large redundancy, because the whole dual codebook must be stored previously and is involved in the calculation as x'_{ij} being the j-th bit position of the i-th dual codeword.
With this decoder we will compute the MAP output for the corresponding Chase decoder reliability value and arrange the points in a scatterplot. For the simulation a (32,21,6) eBCH code was used and $E_b/N_0 = 2$ dB. The number of LRBPs was fixed to $p = 4$. The result is plotted in Figure 4.3.
At first glance it is noticeable that there are two peaks at $\mathrm{L}^{\mathrm{E,Chase}} = \pm 1$. This comes from the fact that $\beta = 1$ in our example.
When looking at the other points, which stem from the update rule (3.17), one can see that the absolute value of $\mathrm{L}^{\mathrm{E,Chase}}$ is generally larger than the corresponding absolute value of $\mathrm{L}^{\mathrm{E,MAP}}$, i.e., the Chase decoder tends to overestimate the extrinsic output.
The dashed line indicates the case where the Chase delivers the same output as MAP. The goal of the postprocessing function f_{pp} must be to attenuate the $\mathrm{L}^{\mathrm{E,Chase}}$ values in such a way, so that they are projected approximately to this line. To approximate f_{pp} we start with sampling a large number of extrinsic output pairs for different SNR values, since the scatterplot changes with the channel noise variance σ^2_{CH}. The goal is to find for a given BCH code and given number p of LRBPs a function f_{pp} dependent on σ^2_{CH}, so that

$$\mathrm{L}^{\mathrm{E,MAP}} \approx f_{\mathrm{pp}}(\mathrm{L}^{\mathrm{E,Chase}}, \sigma^2_{\mathrm{CH}}). \tag{4.3}$$

For each noise level, we want to interpolate the samples to a curve, which indicates where most of the MAP values will lie for a given Chase output. We manipulate the sample set as follows:

- Eliminate all disturbing samples where $\mathrm{L}^{\mathrm{E,Chase}} = \pm\beta$.
- Quantize all $\mathrm{L}^{\mathrm{E,Chase}}$ values with an accuracy considering that there are enough samples in each quantization level to build an expressive mean.
- For each quantized $\mathrm{L}^{\mathrm{E,Chase}}$ value, compute the mean of all corresponding $\mathrm{L}^{\mathrm{E,MAP}}$ samples.

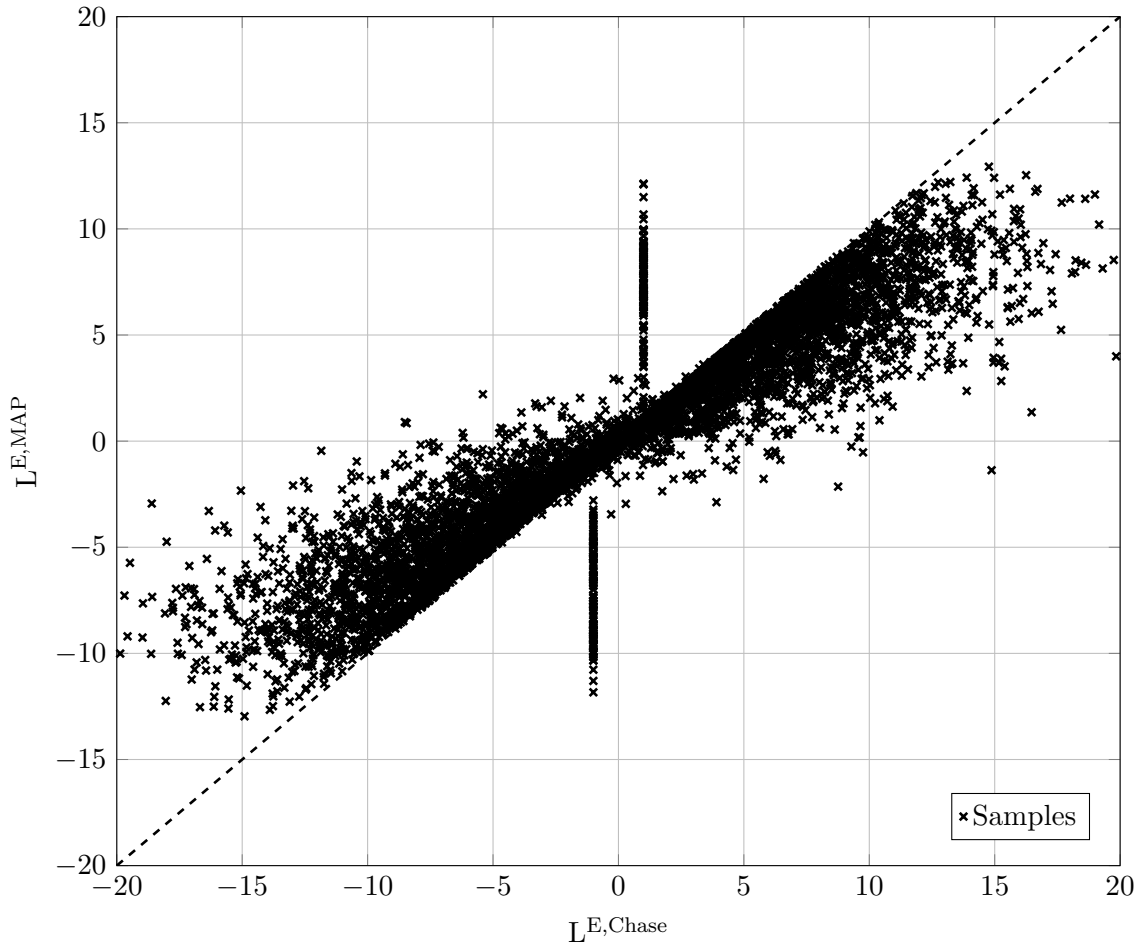


Figure 4.3.: Scatterplot showing the relation of extrinsic information from a Chase compared to a MAP decoder for a (32,21,6) eBCH code with $p = 4$ and $E_b/N_0 = 2$ dB.

- The mean is only computed if the number of $\mathrm{L}^{\mathrm{E,MAP}}$ samples is above a chosen threshold in order to get a reliable mapping.

The resulting interpolated curves are plotted in Figure 4.4. One can observe that especially for lower noise levels, the Chase decoder yields highly overestimated $\mathrm{L}^{\mathrm{E,Chase}}$ values. In the next step, we want to find a function in terms of the noise level, which approximates the mean curves in Figure 4.4. Their shapes indicate a non-linear behavior and show characteristics of a stretched tanh-function.
Therefore, we decide to use this type of function in the following and concentrate on a fixed SNR for deriving scaling factors for stretching the function in both directions, i.e.,

$$\mathrm{L}^{\mathrm{E,MAP}} \approx \gamma \cdot \tanh(\delta \cdot \mathrm{L}^{\mathrm{E,Chase}}). \tag{4.4}$$

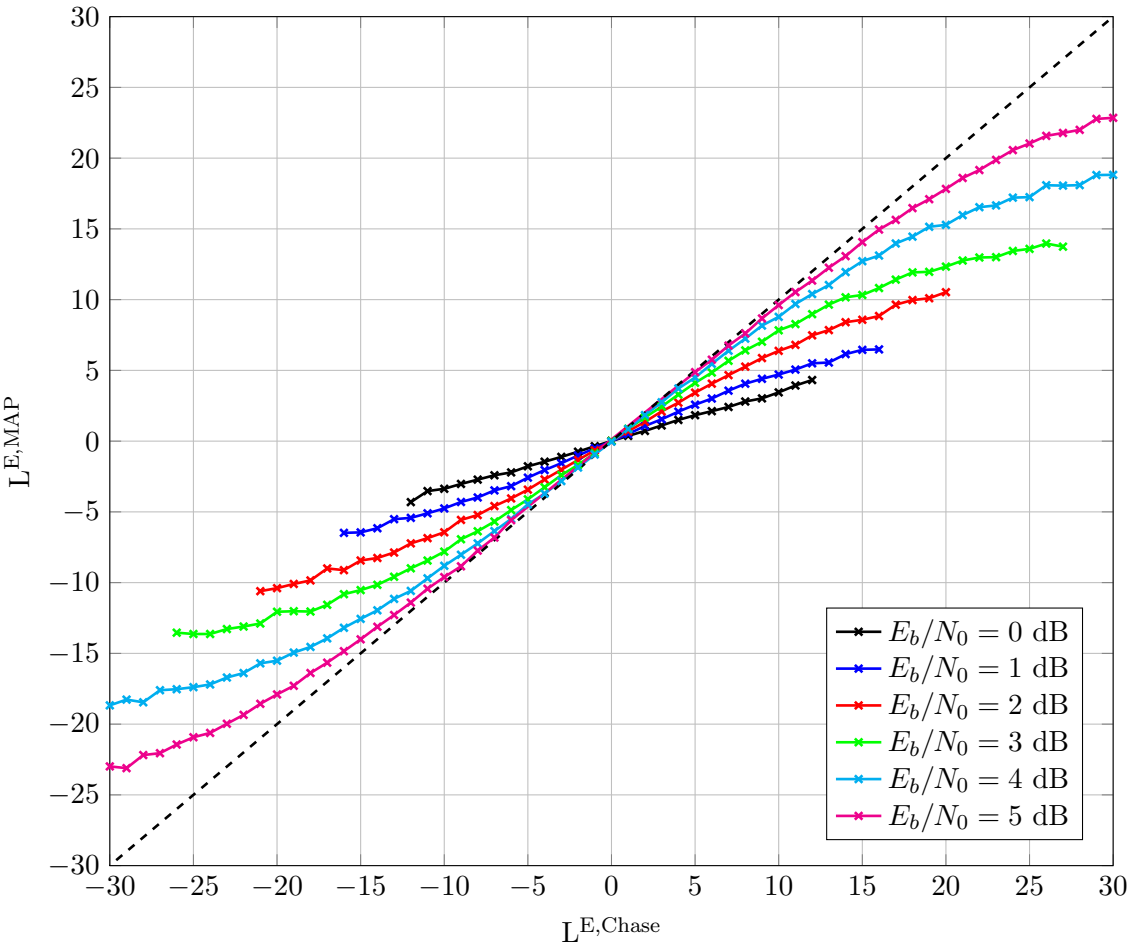


Figure 4.4.: Interpolation of MAP extrinsic LLRs for given Chase decoder outputs at different noise levels for a simulated (32,21,6) eBCH code with $p = 4$.

We can set up an optimization problem for each simulated SNR based on the availabe sample sets as

$$\min_{\gamma,\delta} ||\mathbf{L}^{\mathrm{E,MAP}} - \gamma \cdot \tanh(\delta \cdot \mathbf{L}^{\mathrm{E,Chase}})||_2^2 \qquad \text{s.t.} \quad \gamma, \delta \geq 0. \tag{4.5}$$

This problem can be given into a numerical solver, which delivers values for γ and δ approximating the interpolated curves in Figure 4.4. In the iterative decoding process of a product code it is crucial to use γ and δ dependent on the current SNR level. Since only for the first half-iteration the noise variance of the decoder input corresponds to the channel noise variance σ_{CH}^2, we have to use a different measure to determine γ and δ in subsequent iterations.

According to [35, Sec. 6.4], the mutual information between L^{CH} and the bi-AWGN channel input X is given by the so called J-function, where

$$\begin{aligned}\mathrm{J}\left(\sigma_{\mathrm{CH}}\right) &= \mathrm{I}(\mathrm{L}^{\mathrm{CH}}; X) \\ &= 1 - \frac{1}{\sqrt{2\pi\sigma_{\mathrm{CH}}^2}} \int_{-\infty}^{+\infty} \exp\left(-\frac{\left(z - \sigma_{\mathrm{CH}}^2/2\right)^2}{2\sigma_{\mathrm{CH}}^2}\right) \log(1+\exp(-z))\mathrm{d}z. \end{aligned} \tag{4.6}$$

In this thesis, we use the numerical approximation of the J-function in [57]. Now, $\mathrm{J}\left(\sigma_{\mathrm{CH}}\right)$ can be used to create a mapping between the simulated E_b/N_0 levels and the mutual information. We start by converting from the logarithmic to the linear domain by

$$E_b/N_0|_{\mathrm{lin}} = 10^{0.1 \cdot E_b/N_0|_{\mathrm{dB}}}. \tag{4.7}$$

Next we compute the channel noise variance with respect to the rate of the stand-alone BCH code used in the sample generation by

$$\sigma_{\mathrm{CH}}^2 = \frac{1}{2 \cdot R_{\mathrm{BCH}} \cdot E_b/N_0|_{\mathrm{lin}}}. \tag{4.8}$$

Now we exploit the Gaussian approximation of the channel LLRs, i.e., $\mathrm{L}^{\mathrm{CH}} \sim \mathcal{N}(\frac{2}{\sigma_{\mathrm{CH}}^2}, \frac{4}{\sigma_{\mathrm{CH}}^2})$, which simplifies the standard deviation σ' of the model to

$$\sigma' = \sqrt{\frac{4}{\sigma_{\mathrm{CH}}^2}} = \frac{2}{\sigma_{\mathrm{CH}}}. \tag{4.9}$$

In the last step, we plug this value for σ' into the J-function to obtain

$$\mathrm{I}(\mathrm{L}^{\mathrm{CH}}; X) = \mathrm{J}(\sigma'). \tag{4.10}$$

In Table 4.1 the mutual information values $\mathrm{MI} = \mathrm{I}(\mathrm{L}^{\mathrm{CH}}; X)$ for the corresponding E_b/N_0 ratios are given for the simulated (32,21,6) eBCH code. Furthermore, the tanh scaling factors γ and δ, which are the solution to the optimization problem (4.5), are provided. To avoid the usage of such a look-up table in the construction of the postprocessing function, we try to derive a functional dependency from γ to MI and from δ to MI, respectively. When plotting MI against the outer tanh scaling factor γ we can observe an exponential growth for increasing MI. For the inner factor δ the evolution for increasing MI can be approximately modelled by a falling linear function. Thus, we can apply a simple exponential/ linear regression to get functions for γ and δ in terms of MI and formulate the final postprocessing function for Chase decoding of a (32,21,6) eBCH code

$E_b/N_0\|_{\mathrm{dB}}$	$E_b/N_0\|_{\mathrm{lin}}$	σ^2_{CH}	$\mathrm{MI} = \mathrm{I}(\mathrm{L}^{\mathrm{CH}}; X)$	γ	δ
-6	0.25	3.03	0.20	0.01	0.623
-3	0.50	1.50	0.35	0.53	0.127
0	1	0.76	0.57	5.82	0.067
0.5	1.12	0.68	0.61	6.50	0.074
1	1.25	0.60	0.65	7.84	0.070
1.5	1.41	0.54	0.69	10.4	0.059
2	1.58	0.48	0.73	12.1	0.058
2.5	1.77	0.42	0.77	12.6	0.057
3	2.00	0.38	0.80	14.8	0.056
3.5	2.23	0.34	0.84	17.7	0.052
4	2.51	0.30	0.86	20.2	0.048
4.5	2.81	0.27	0.89	23.7	0.043
5	3.16	0.24	0.91	28.5	0.036

Table 4.1.: Parameters of the postprocessing function for Chase decoding of a (32,21,6) eBCH code with $p = 4$.

with $p = 4$ LRBPs as

$$\begin{aligned} \mathrm{L}^{\mathrm{E,MAP}} &= f_{\mathrm{pp}}(\mathrm{L}^{\mathrm{E,Chase}}, \mathrm{MI}) = \gamma\,(\mathrm{MI}) \cdot \tanh(\delta\,(\mathrm{MI}) \cdot \mathrm{L}^{\mathrm{E,Chase}}) \\ \text{where} \quad \gamma &= 0.2816 \cdot \exp(4.9651 \cdot \mathrm{MI}) \quad \text{and} \quad \delta = -0.1031 \cdot \mathrm{MI} + 0.1377. \end{aligned} \tag{4.11}$$

In Figure 4.5 we check how well the postprocessing function f_{pp} from (4.11) approximates the interpolated sample curves from Figure 4.4. We can observe that the derived scaling of the tanh function evaluated for different mutual information values delivers a good approximation of the mean curves. Simulations of iterative TPD show that exactly this range $0.5 < \mathrm{MI} < 1$ is the region where very strict attenuation of $\mathrm{L}^{\mathrm{E,Chase}}$ is crucial in order to avoid massive over-estimation by the Chase decoder.
Due to this fact and the aim to keep the postprocessing function within a reasonable complexity, the function in (4.11) can be considered as good enough for an application in an iterative process.
Analogous to the procedure explained in this section, the postprocessing functions of the form

$$f_{\mathrm{pp}}(\mathrm{L}^{\mathrm{E,Chase}}, \mathrm{MI}) = \gamma\,(\mathrm{MI}) \cdot \tanh(\delta\,(\mathrm{MI}) \cdot \mathrm{L}^{\mathrm{E,Chase}}) \tag{4.12}$$

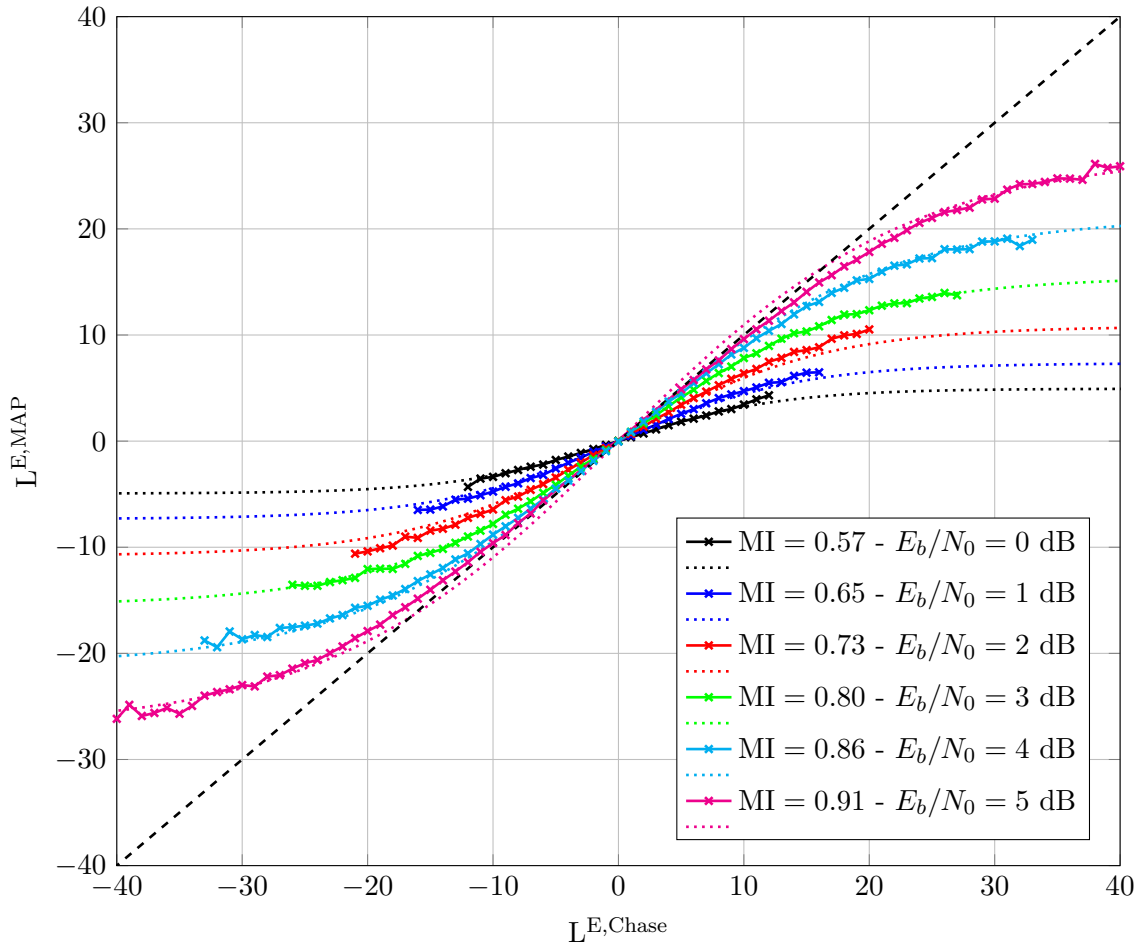


Figure 4.5.: Postprocessing function (dotted) approximating the interpolated sample curves (solid) of MAP extrinsic LLRs for a corresponding Chase output value for the (32,21,6) eBCH code with $p = 4$.

were derived for other codelengths and different number p of LRBPs in the Chase decoding process and are summarized in Table 4.2.

Code	p	γ	δ
(16,7,6)	4	$2.7587 \cdot \exp(2.8504 \cdot \mathrm{MI})$	$-0.0910 \cdot \mathrm{MI} + 0.1129$
(16,7,6)	5	$2.5327 \cdot \exp(2.9266 \cdot \mathrm{MI})$	$-0.1516 \cdot \mathrm{MI} + 0.1601$
(16,11,4)	4	$0.1269 \cdot \exp(6.4053 \cdot \mathrm{MI})$	$-0.3092 \cdot \mathrm{MI} + 0.3436$
(16,11,4)	5	$0.0604 \cdot \exp(6.4053 \cdot \mathrm{MI})$	$-0.3888 \cdot \mathrm{MI} + 0.4164$
(32,21,6)	4	$0.2816 \cdot \exp(4.9651 \cdot \mathrm{MI})$	$-0.1031 \cdot \mathrm{MI} + 0.1377$
(32,21,6)	5	$0.3772 \cdot \exp(4.6686 \cdot \mathrm{MI})$	$-0.1183 \cdot \mathrm{MI} + 0.1490$
(32,26,4)	4	$0.0128 \cdot \exp(7.4994 \cdot \mathrm{MI})$	$-0.3096 \cdot \mathrm{MI} + 0.3565$
(32,26,4)	5	$0.0248 \cdot \exp(6.7793 \cdot \mathrm{MI})$	$-0.2371 \cdot \mathrm{MI} + 0.2894$
(64,51,6)	4	$0.0053 \cdot \exp(8.9008 \cdot \mathrm{MI})$	$-0.0582 \cdot \mathrm{MI} + 0.1020$
(64,51,6)	5	$0.0027 \cdot \exp(9.6331 \cdot \mathrm{MI})$	$-0.3157 \cdot \mathrm{MI} + 0.3368$
(64,57,4)	4	$0.0078 \cdot \exp(8.6356 \cdot \mathrm{MI})$	$-0.3112 \cdot \mathrm{MI} + 0.3687$
(64,57,4)	5	$0.0204 \cdot \exp(6.8104 \cdot \mathrm{MI})$	$-0.2299 \cdot \mathrm{MI} + 0.2908$
(128,113,6)	4	$3.6\text{e-}6 \cdot \exp(16.1547 \cdot \mathrm{MI})$	$-0.3048 \cdot \mathrm{MI} + 0.3392$
(128,113,6)	5	$5.2\text{e-}6 \cdot \exp(15.7214 \cdot \mathrm{MI})$	$-0.3242 \cdot \mathrm{MI} + 0.3592$
(128,120,4)	4	$2.7\text{e-}7 \cdot \exp(17.7988 \cdot \mathrm{MI})$	$-1.5507 \cdot \mathrm{MI} + 1.6064$
(128,120,4)	5	$2.4\text{e-}7 \cdot \exp(18.1149 \cdot \mathrm{MI})$	$-1.5112 \cdot \mathrm{MI} + 1.5549$

Table 4.2.: Postprocessing functions for Chase-Pyndiah decoding.

4.2.2. Application of Postprocessing in the Iterative Process

In this subsection we want to apply the derived postprocessing functions f_{pp} from Table 4.2 to product codes in the iterative Ch-Py TPD process. If we refer to MI in the following, we refer to the mutual information between the decoder input L^{IN} and channel input X, i.e.,

$$\text{MI} = \text{I}\left(\text{L}^{\text{IN}}, X\right), \tag{4.13}$$

because f_{pp} was derived based on the decoder input. For the first half-iteration we have $\text{L}^{\text{IN}} = \text{L}^{\text{CH}}$, but we have not defined how to obtain the correct coefficients of f_{pp} in terms of the mutual information MI. We present two different options. The first way corresponds to an instantaneous estimation of MI in every half-iteration, where we get the possibility of being able to dynamically react on rising and falling MI values with proper adjustment of f_{pp}. Due to the increase in computational complexity compared to standard Ch-Py decoding from Algorithm 1, a second way is proposed, where we store the average MI values for a fixed SNR and use them to define the iteration-dependent expected coefficients of f_{pp} in advance. Finally both options are compared to each other and to the performance of standard Ch-Py decoding, respectively.

Instantaneous Measurement of the Mutual Information

Only for the first half-iteration, the choice of MI is clear, assuming the channel noise variance σ_{CH}^2 is known. Therefore, we can initialize MI by

$$\text{MI} = \text{J}\left(\frac{2}{\sigma_{\text{CH}}}\right). \tag{4.14}$$

However, for subsequent iterations, the decoder input LLR L^{IN} is not distributed according to a Gaussian distribution with variance $4/\sigma_{\text{CH}}^2$ anymore. In [58], a method on estimating the mutual information based on the absolute values of the LLR is shown. It holds under the assumption of ergodicity, i.e., the samples are representative for the whole stochastic model. Therefore, even for non-Gaussian or unknown distributions, mutual information can be approximated as

$$\text{I}(\text{L}; X) = 1 - \text{E}\left[\log_2\left(1 + e^{-\text{L}}\right)\right] \approx 1 - \frac{1}{n}\sum_{i=1}^{n} \log_2\left(1 + e^{-x_i \cdot \text{L}_i}\right). \tag{4.15}$$

Since we do not know the correct data $\boldsymbol{x}$, we only eploit the magnitudes of the LLRs L and measure the error probability by

$$P_{e_i} = \frac{e^{+|\mathrm{L}_i|/2}}{e^{+|\mathrm{L}_i|/2} + e^{-|\mathrm{L}_i|/2}}. \tag{4.16}$$

Thus we may estimate the mutual information by

$$\mathrm{I}(\mathrm{L}; X) \approx 1 - \frac{1}{n}\sum_{i=1}^{n} \mathrm{H}_2\left(P_{e_i}\right) = 1 - \frac{1}{n}\sum_{i=1}^{n} \mathrm{H}_2\left(\frac{e^{+|\mathrm{L}_i|/2}}{e^{+|\mathrm{L}_i|/2} + e^{-|\mathrm{L}_i|/2}}\right) \tag{4.17}$$

in the following, where H_2 is the binary entropy function (2.3).
The equivalence of (4.10) and (4.17) can be easily checked by uniformly sampling a large number of channel input symbols X according to the bi-AWGN channel alphabet, i.e.,

$$X \sim \mathcal{U}\left\{\pm 1\right\}. \tag{4.18}$$

For each symbol, we generate noise Z, which is normal Gaussian distributed with channel noise variance σ^2_{CH}. Thus we may write

$$Z \sim \mathcal{N}\left(0, \sigma^2_{\mathrm{CH}}\right). \tag{4.19}$$

The channel output is simply given by

$$Y = X + Z \tag{4.20}$$

and the corresponding channel LLRs are computed as

$$\mathrm{L}^{\mathrm{CH}} = \frac{2Y}{\sigma^2_{\mathrm{CH}}}. \tag{4.21}$$

Then it holds, that

$$\mathrm{I}(\mathrm{L}^{\mathrm{CH}}; X) = \mathrm{J}\left(\frac{2}{\sigma_{\mathrm{CH}}}\right) \approx 1 - \frac{1}{n}\sum_{i=1}^{n} \mathrm{H}_2\left(\frac{e^{+|\mathrm{L}_i^{\mathrm{CH}}|/2}}{e^{+|\mathrm{L}_i^{\mathrm{CH}}|/2} + e^{-|\mathrm{L}_i^{\mathrm{CH}}|/2}}\right). \tag{4.22}$$

Since the coefficients of f_{pp} were derived with respect to MI, resembling the mutual information at the decoder input, we need to plug $\mathrm{L}^{\mathrm{IN}} = \mathrm{L}^{\mathrm{CH}} + \mathrm{L}^{\mathrm{A}}$ into (4.17) to estimate MI. Because of the fact that the a priori information in iteration I corresponds to the postprocessed extrinsic information from the previous half-iteration $I-1$, i.e., $\mathrm{L}^{\mathrm{A},I} = \mathrm{L}^{\mathrm{E,pp},I-1}$, we avoid measuring the mutual information MI based on the overestimated

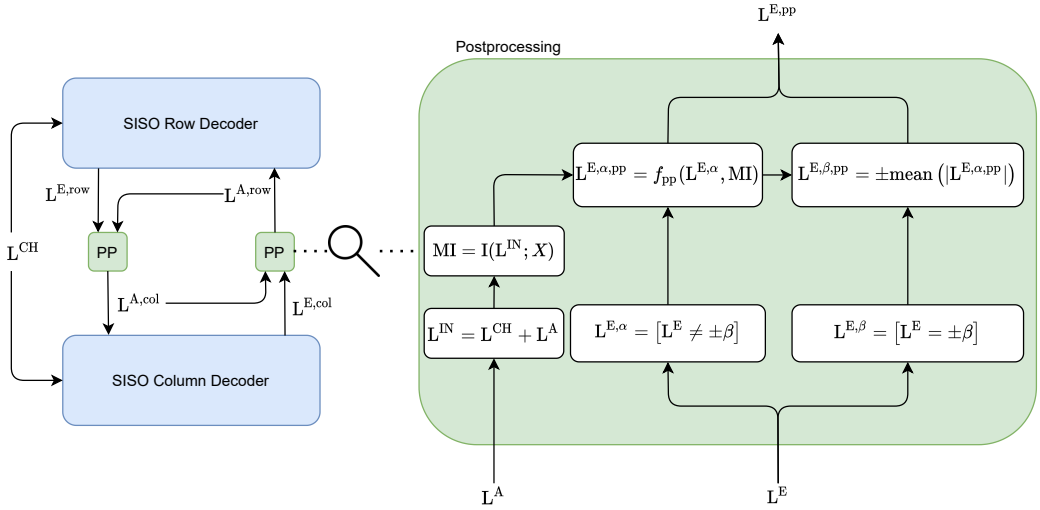


Figure 4.6.: Chase-Pyndiah decoding with extrinsic information postprocessing.

extrinsic values, which directly come out of the Chase decoder. However, f_{pp} cannot be applied in a straightforward manner. The proposed procedure of postprocessing is shown in the green block in Figure 4.6, where we also determine how to deal with the case that there is no competing codeword, i.e., $\mathrm{L}^{\mathrm{E}} = \pm\beta$. For certain choices of component codes and of p, respectively, the number of these cases may be more than half of all L^{E} values and therefore has a powerful impact on the algorithm performance.

Pyndiah used the trick of a normalization from (3.23) in order to set the mean of all $\mathrm{L}^{\mathrm{E}} \neq \beta$ values to 1. This leads to a balance between $\mathrm{L}^{\mathrm{E}} = \beta$ and values where $\mathrm{L}^{\mathrm{E}} \neq \beta$, so that the magnitudes do not loose proportion among each other.

In this chapter, we give up this normalization because the idea of approximating an optimal MAP output with f_{pp} will make this operation obsolete.

Now, a natural approach would be to choose the magnitude of the β values by orientating on the other, already postprocessed, extrinsic information values. In other words, we split L^{E} in two groups. The first group is called $\mathbf{L}^{\mathrm{E},\alpha}$ and obtained from rule (3.17) and the second group $\mathbf{L}^{\mathrm{E},\beta}$ stems from rule (3.18), respectively. Then we plug each value $\mathrm{L}^{\mathrm{E},\alpha}$ into the postprocessing function f_{pp} to get the postprocessed extrinsic information values $\mathrm{L}^{\mathrm{E},\alpha,\mathrm{pp}}$ as

$$\mathrm{L}^{\mathrm{E},\alpha,\mathrm{pp}} = f_{\text{pp}}\left(\mathrm{L}^{\mathrm{E},\alpha}, \mathrm{MI}\right). \tag{4.23}$$

The missing $\mathrm{L}^{\mathrm{E},\beta,\mathrm{pp}}$ values are then simply calculated as

$$\mathrm{L}^{\mathrm{E},\beta,\mathrm{pp}} = \text{sign}\left(\mathrm{L}^{\mathrm{E},\beta}\right) \cdot \left|\text{mean}\left(\mathbf{L}^{\mathrm{E},\alpha,\mathrm{pp}}\right)\right|. \tag{4.24}$$

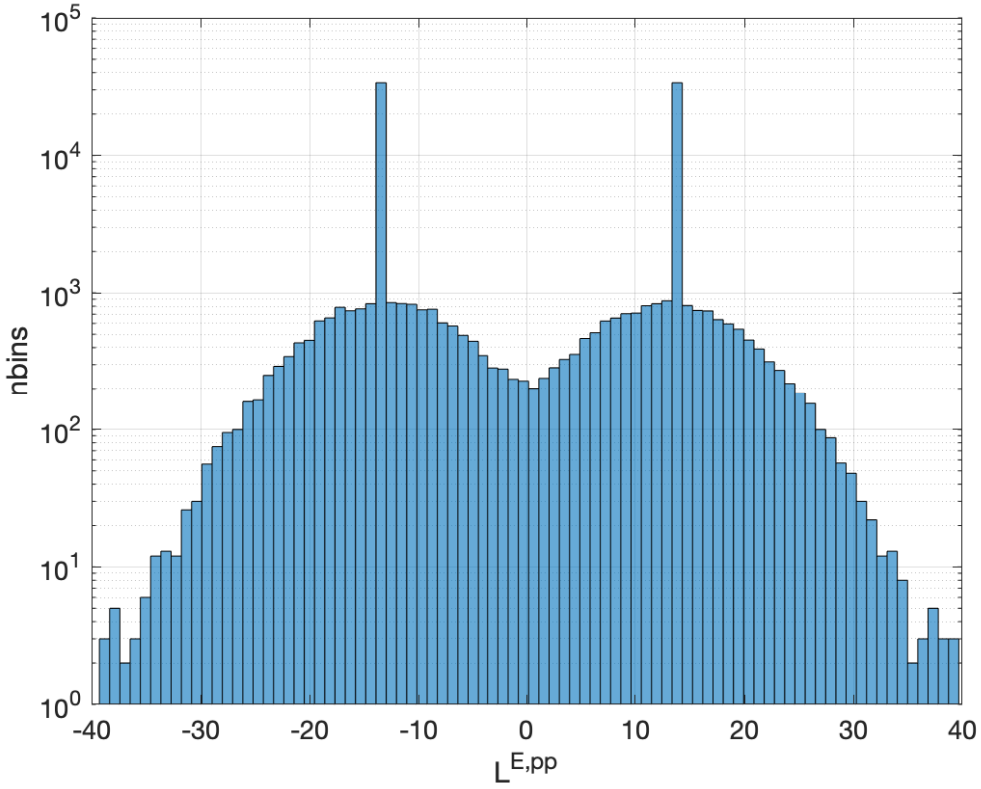


Figure 4.7.: Choosing β as the mean absolute value of the postprocessed extrinsic information values.

This choice can be very well graphically justified by the histogram in Figure 4.7 when regarding a large number of samples of $\mathbf{L}^{\mathrm{E,pp}}$. When transmitting over the bi-AWGN channel, the distribution of all postprocessed $\mathbf{L}^{\mathrm{E},\alpha,\mathrm{pp}}$ approximately has a shape of the sum of two Gaussian curves. The absolute mean value of each of these curves is approximately equal because we assume a uniformly distributed input and deal with a symmetric Ch-Py decoder and a symmetric postprocessing function. The β values are now the conditional mean estimate of $\mathbf{L}^{\mathrm{E},\alpha,\mathrm{pp}}$, which results in two peaks on top of the Gaussian curves in the histogram. After that, all $\mathbf{L}^{\mathrm{E,pp}}$ together are passed as new a-priori information $\mathbf{L}^{\mathrm{A}}$ to the other component decoder for the next half-iteration. The procedure from Figure 4.6 is repeated until the maximum number of iterations $I_{\max}$ is reached. Whereas the final binary decision $\hat{\boldsymbol{C}}$ in Algorithm 1 and 2 is taken on the most likely codewords $\boldsymbol{D}$ of the Chase decoder, we approximate now true APPs and estimate $\hat{\boldsymbol{C}}$ based on the HD on

$$\mathrm{L}^{\mathrm{APP}} = \mathrm{L}^{\mathrm{CH}} + \mathrm{L}^{\mathrm{A}} + \mathrm{L}^{\mathrm{E,pp}}. \tag{4.25}$$

We may summarize the essential steps of this modified version of Ch-Py decoding as follows.

ALGORITHM 3

Chase-Pyndiah Decoding with Extrinsic Information Postprocessing Based on Estimated Mutual Information

- Chase decoder with p LRBPs and BDD decoding of test list.
- Instantaneous estimation of $\mathrm{I}\left(\mathrm{L}^{\mathrm{IN}}; X\right)$ by (4.17) in each half-iteration.
- Postprocessing of $\mathrm{L}^{\mathrm{E},\alpha}$ according to (4.23) based on f_{pp} from Table 4.2.
- Postprocessing of $\mathrm{L}^{\mathrm{E},\beta}$ according to (4.24).
- Final binary decision based on $\mathrm{L}^{\mathrm{APP}}$ (4.25).

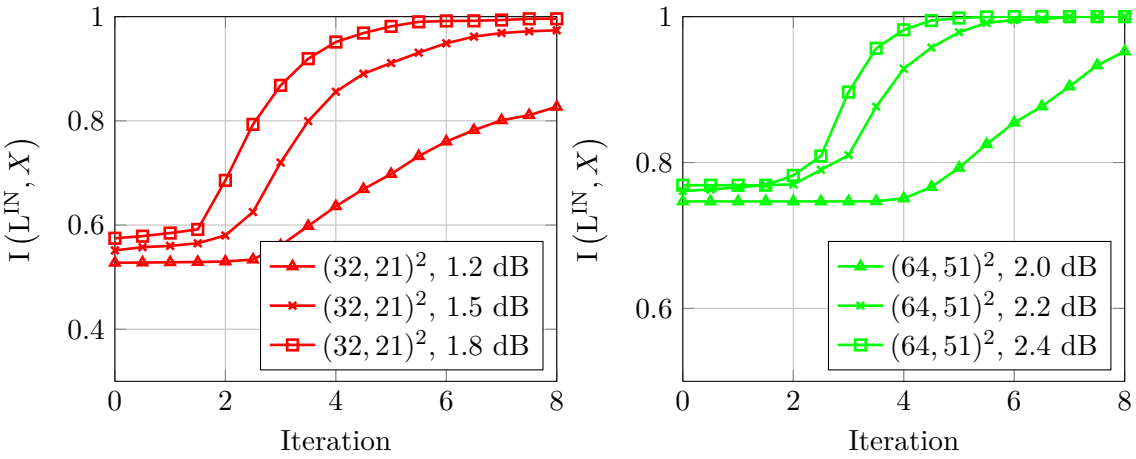


Figure 4.8.: Evolution of the average estimated mutual information over the decoding half-iterations for two different product codes at different SNRs decoded with Chase-Pyndiah decoding ($p = 4$) with postprocessing.

Postprocessing Based on Predefined Coefficients

When comparing the complexity of Algorithm 3 to the one of standard Ch-Py decoding in Algorithm 1, we can observe that the number of arithmetic scaling operations are roughly the same for both variants of the algorithm, apart from the estimation of MI. The postprocessing of each $\mathrm{L}^{\mathrm{E},\alpha}$ value approximately results in the same effort as the multiplication with α, when f_{pp} is efficiently implemented, e.g., in terms of a look-up-table. Furthermore, in both Chase-Pyndiah variants, the classical one and the one with postprocessing, the mean of all $\mathrm{L}^{\mathrm{E},\alpha}$ values has to be computed. In standard Ch-Py (Alg. 1), this mean value is required to apply the normalization equation (3.23) and in Ch-Py with postprocessing (Alg. 3) we use it as the "new" β value.
Consequently, only the estimation of the mutual information in each half-iteration causes an increase in complexity. However, we may find a solution to this if we decide for using previously stored iteration-dependent values of MI for a fixed code and a fixed SNR.
To obtain these predefined values of MI, we simulate over the bi-AWGN channel the transmission of a very large number of blocks, where each one is decoded via Ch-Py decoding with postprocessing and instantaneous mutual information measurement according to Algorithm 3. In each half-iteration, we store MI and build the average value after that. We do not apply early stopping in order to capture the complete evolution to one. The average mutual information values for the $(32, 21, 6)^2$ and the $(64, 51, 6)^2$ product codes at different SNRs are plotted over 16 half-iterations in Figure 4.8. From the evolution of the curves, we can see that the average mutual information in the first iterations of the Ch-Py decoder almost remains constant at lower SNR before it starts

converging towards one. This behavior indeed only represents the *average* value of MI, because actually the value is alternating, which comes from the suboptimal nature of the Ch-Py algorithm and the changing LRBPs for which the test list is built on in each half-iteration. Thus, it may happen that the decoder needs some iterations until it has found the "right track". Interestingly, the evolution of MI is similar to the one captured by the extrinsic information transfer (EXIT) charts in [59], although the authors came up with a different method to derive a scaling factor α.
The idea to store the values from Figure 4.8 and invoke them in the postprocessing block in Figure 4.6 can be formalized by a slight modification of Algorithm 3, resulting in Algorithm 4.

ALGORITHM 4

Chase-Pyndiah Decoding with Extrinsic Information Postprocessing Based on Predefined Average Mutual Information Values

- Chase decoder with p LRBPs and BDD decoding of test list.
- Predefined iteration-dependent values of $\mathrm{I}\left(\mathrm{L}^{\mathrm{IN}}; X\right)$ as in Figure 4.8.
- Postprocessing of $\mathrm{L}^{\mathrm{E},\alpha}$ according to (4.23) based on f_{pp} from Table 4.2.
- Postprocessing of $\mathrm{L}^{\mathrm{E},\beta}$ according to (4.24).
- Final binary decision based on $\mathrm{L}^{\mathrm{APP}}$ (4.25).

Algorithm 4 thus is indeed suitable for practical applications where we want to achieve a certain target BER and may fix the SNR, therefore. How much we loose in terms of performance will be evaluated in the following.

Simulation Results

In Figure 4.9 the BER performance curves for the $(32, 21, 6)^2$ (red), $(64, 51, 6)^2$ (green) and $(128, 113, 6)^2$ (brown) product code are plotted. We want to compare standard Ch-Py decoding (Alg. 1 - dotted with circles) to Ch-Py decoding with extrinsic information postprocessing and estimating the mutual information in each half-iteration (Alg. 3 - solid with squares). On top of that, we provide the results for Chase-Pyndiah decoding with extrinsic information postprocessing based on the fixed and predefined mutual information values from Figure 4.8 (Alg. 4 - solid with diamonds).
For the $(32, 21, 6)^2$ and $(64, 51, 6)^2$ code, a further reference is given in form of two curves corresponding to decoding all rows and columns of the product codes iteratively with BCJR (dashed with triangle).

All versions of Ch-Py decoding assumed a Chase decoder with $p = 4$ LRBPs and every algorithm has been run for $I = 8$ iterations. The results of Algorithm 3 outperform 1 in terms of BER for all simulated codes. For the $(32, 21, 6)^2$ product code there is a constant gain of approximately 0.1 dB over the whole SNR range, whereas for the $(64, 51, 6)^2$ the gain vanishes for increasing SNR. The $(128, 113, 6)^2$ code however does not show any sign of decreasing gain.

It is not clear why the gain for the $(64, 51, 6)^2$ product code is not constant across all SNRs. It may be explained by the fact that Pyndiah provided fixed scaling factors in combination with normalization for a large set of different codes. Therefore, these scaling coefficients are not optimal and there are different gains for different component codes.

We can estimate a trend of an increasing gap between all versions of Ch-Py decoding and iterative BCJR decoding for increasing blocklength. The reason for this is the accuracy of the exchanged extrinsic information between the component decoders. For larger codelengths, the suboptimal Chase output delivers extrinsic information values, which are too far away from the true MAP value. Clearly, $p = 4$ and a resulting Chase decoder test list with $2^p = 16$ test words, is not able to capture all the closest codewords, which contribute to the extrinsic information value the most. But setting the number of LRBPs to $p = 5$ is a good trade-off between a moderate increase in complexity and benefit in BER performance.

Finally, we also want to compare both versions of postprocessing (Alg. 3 and Alg. 4) with each other, i.e, instantaneously estimating MI versus predefined MI. As expected Algorithm 3 performs better than Algorithm 4. However, the gap between the curves is tiny for larger blocklengths. Therefore it might be advisible to use this option of predefined MI values and save the calculation of (4.17) in each half-iteration. Especially in practical systems, where a fixed target BER wants to be achieved and the main scope is on high throughputs, the very small penalty in SNR is negligible compared to the reduction in complexity.

Note that in Figure 4.9 only product codes with $t = 2$ error correcting eBCH component codes are investigated. As already noticed in Section 4.1 with Algorithm 2 based on the fraction of valid codewords, it is hard to enhance the performance of product codes consisting of extended Hamming component codes ($t = 1$) decoded with standard Ch-Py decoding. This phenomenon appears for Ch-Py decoding with extrinsic information postprocessing, too. In general, approximately the same BERs can be achieved by using the postprocessing functions from Table 4.2. This confirms the hypothesis that product codes with Hamming component codes are quite robust concerning scaling and the need of normalization [51, 53].

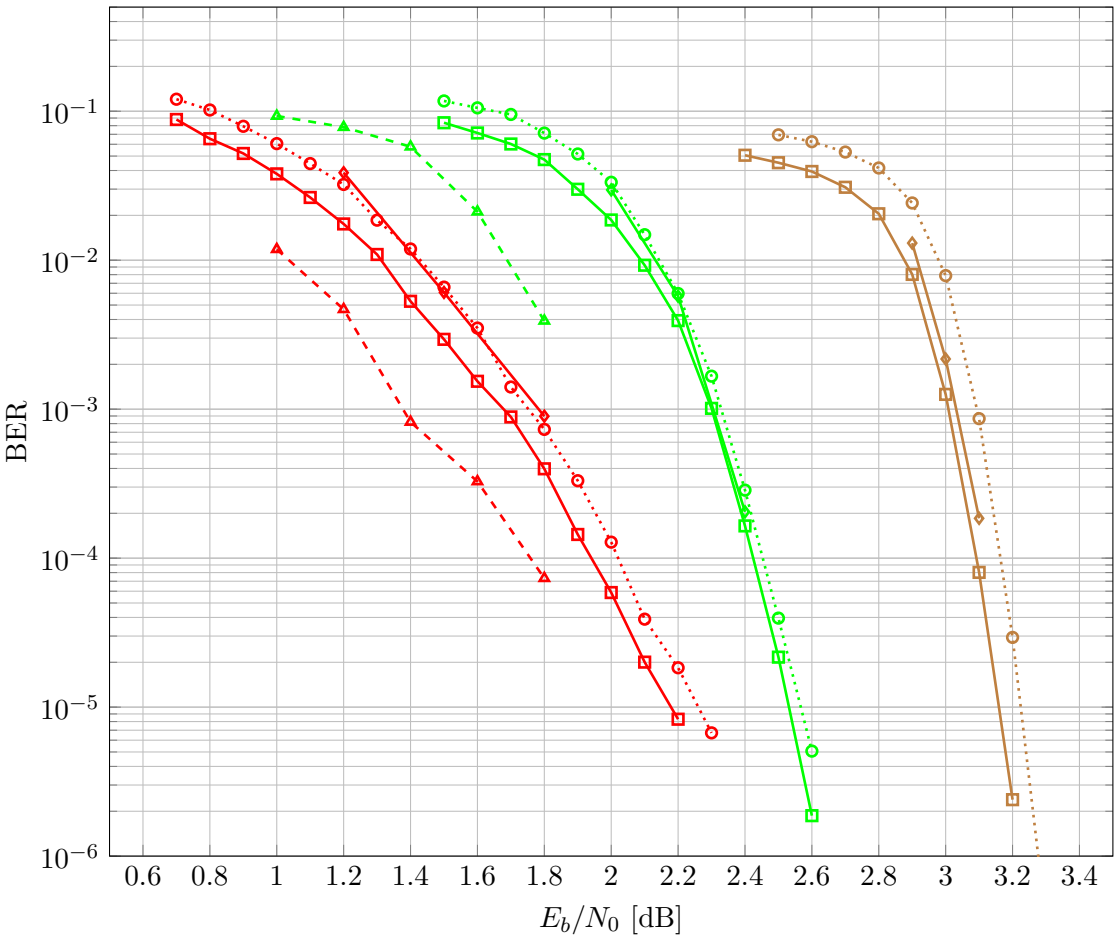


Figure 4.9.: Simulation results of product codes decoded with $I = 8$ iterations of different versions of Chase-Pyndiah decoding with $p = 4$ compared to $I = 8$ iterations of iterative BCJR decoding.

5. Asymptotic Decoding Analysis of Chase-Pyndiah Decoding

In Section 3.2 we have seen that product codes can be regarded as an instance of GLDPC codes. Moreover, we can assign a product code to a certain GLDPC code ensemble $\mathcal{G} = (\mathcal{C}, M)$ determined by the component code $\mathcal{C}$ and the number of CNs M.
For such GLDPC code ensembles it is possible to measure how the extrinsic message distribution evolves with increasing number of iterations for a large number of VNs N. This procedure called density evolution (DE) analysis can be used to determine the iterative decoding threshold of the ensemble. The threshold denotes the SNR at which the BER tends to zero with probability approaching one in the asymptotic limit of large block length ($N \to \infty$). Since the number of VNs N is diretly determined by the number of CNs M and the component code $\mathcal{C}$, we can analyze this asymptotic behavior of an infinitely long GLDPC code representative for the product code belonging to the ensemble, in the following.
Instead of MAP decoding at the CNs as proposed in [37], we want to investigate the performance of using the Chase algorithm. However, for asymptotic analysis like DE, it is essential to assume the exchange of purely extrinsic information.
In Section 5.1 we will see that standard Ch-Py decoding (Alg. 1) is intrinsic. To overcome this problem, we will introduce a new extrinsic version of Ch-Py decoding. Due to the fact that Pyndiah's scaling coefficients in (3.21) and (3.22) are only suited for standard Ch-Py (Alg. 1), we derive postprocessing functions for the extrinsic decoder in Section 5.2. Extrinsic Ch-Py decoding comes with a huge increase in complexity and has the only purpose to make DE analysis in Section 5.3 possible, but due to the still suboptimal nature of Ch-Pyit is not possible to derive an analytical expression for the error probability of the decoder. Instead a Monte-Carlo (MC) method is proposed in Section 5.3.

5.1. An Extrinsic Version of Chase-Pyndiah Decoding

5.1.1. The Intrinsic Nature of a Chase Decoder

Standard Ch-Py TPD as originally introduced in [5] is intrinsic due to the following reasons:

- The Tanner graph of a product code has girth 8, i.e. it contains cycles. This means that at some point in the iterative process a CN is fed with a priori information from VN i, which contains already known information about the i-th VN.

- The Chase decoder output is computed based on the closest codeword $\boldsymbol{d}$ in the candidate list. This candidate list is again based on decoding the testlist, which is formed by flipping the LRBPs. When generating the extrinsic information for bit position i, i.e., $\mathrm{L}_i^{\mathrm{E}}$, no a priori information for the i-th bit position is allowed to contribute to this calculation. According to the update rule

$$\mathrm{L}_i^{\mathrm{E}} = \frac{1}{2}\left(\sum_{j\neq i}^{n} \mathrm{L}_j^{\mathrm{IN}}\left(d_j - \bar{c}_j\right)\right) d_i, \tag{5.1}$$

  $\mathrm{L}_i^{\mathrm{IN}}$ is not directly involved in the calculation, but the value of $\mathrm{L}_i^{\mathrm{IN}} = \mathrm{L}_i^{\mathrm{CH}} + \mathrm{L}_i^{\mathrm{A}}$ is directly involved in forming the test list, since the a priori information influences the LRBP. Therefore a priori information of the i-th bit $\mathrm{L}_i^{\mathrm{A}}$ is *indirectly* used for generating $\mathrm{L}_i^{\mathrm{E}}$.

- If there is no competing codeword $\bar{\boldsymbol{c}}$ for the i-th bit position in the candidate list, the update rule for $\mathrm{L}_i^{\mathrm{E}}$ is purely based on the i-th bit position of the Chase decoder decision codeword $\boldsymbol{d}$, i.e.,

$$\mathrm{L}_i^{\mathrm{E}} = \beta d_i. \tag{5.2}$$

  The Chase decoder also has to use this update rule in (5.2), if the candidate list is completely empty. If this is the case, we have that $\boldsymbol{d} = \boldsymbol{r} = \mathrm{HD}(\mathbf{L}^{\mathrm{IN}})$. Then clearly $\mathrm{L}_i^{\mathrm{E}}$ is a *direct* function of $\mathrm{L}_i^{\mathrm{IN}}$.

For finite-length simulations, we can not avoid the impact of cycles in the Tanner graph on the decoding performance. When assuming infinitely long GLDPC codes however, we may neglect this issue.
To convert formulas (5.1) and (5.2) in true extrinsic update rules, the Chase decoder has to be adjusted.

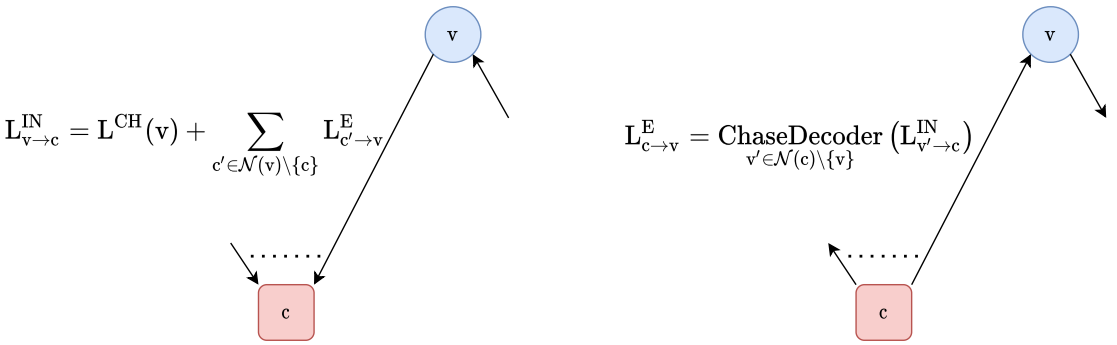


Figure 5.1.: VN and CN update of extrinsic Chase-Pyndiah decoding based on the GLDPC principle.

5.1.2. Chase-Pyndiah Decoding With Extrinsic Check Node Update

In [37] decoding of GLDPC codes was proposed by iterative message passing over the Tanner graph with extrinsic information generated at the CNs. For Gallager's classical LDPC codes, the extrinsic information is computed according to the MAP rule of SPC codes. For GLDPC codes with linear block codes as code constraints, extrinsic information based on the MAP principle can be obtained by performing, e.g., BCJR decoding over the code trellis.
To reduce the decoding complexity, it is also possible to employ a suboptimal decoding algorithm at the CNs. This concept is already commonly used for LDPC codes, where min-sum decoding is another valid method for generating extrinsic information [9, Sec. 5.5]. This idea can be extended to GLDPC codes, where suboptimal Chase decoding might be a lower complexity compromise compared to MAP decoding.
However, it is important that the suboptimal strategy does not violate the extrinsic nature of messages, i.e., the algorithm at CN c computes its extrinsic output for VN v based on all incoming messages into CN c, except from the message from VN v, for which we want to compute the extrinsic information L^{E} for. This concept is visualized in Figure 5.1 as well.

We call this concept extrinsic Ch-Py decoding in the following. The VN update rule for extrinsic Ch-Py decoding and GLDPC codes in general is the same as for standard Ch-Py. If we separately decode at first all rows and after that all columns, or vice versa, the update condition in the VN rule is automatically satisfied.
It remains open how to achieve that $L^{E}_{c\to v}$ is derived without using $L^{IN}_{v\to c}$. To solve this problem, we simply propose to set

$$L^{IN}_{v\to c} = 0, \tag{5.3}$$

i.e., the decoder input LLR $\mathrm{L}_{\mathrm{v}\to\mathrm{c}}^{\mathrm{IN}}$ is set to zero, when the extrinsic information at CN c for VN v is computed. As a consequence, the Chase algorithm has to be run bit-wise, which comes with a massive increase in complexity compared to standard "intrinsic" Ch-Py decoding (Alg. 1), where the extrinsic information is computed row-wise and column-wise, respectively.

The modifications compared to intrinsic Ch-Py though lead to the problem that the scaling factors proposed by Pyndiah in [5], given in (3.21) and (3.22), do not work properly and have to be optimized.

From simulations, it soon became clear that product codes, which consist of Hamming component codes, are not as sensitive concerning the missing normalization, as BCH component codes with higher error correcting capability t. For standard intrinsic Ch-Py decoding it turned out that a constant scaling of $\alpha = 0.5$ and $\beta = 1$ achieves approximately the same performance as Pyndiah's iteration-dependent ones and not considering the normalization [51, 53]. In [24, Sec. 6] repeated trials on simulated performance have also shown that $\alpha = 0.5$ for BCJR and $\alpha = 0.4375$ for Chase decoding are performing best in terms of BER. Unfortunately, it is unclear for which Chase decoder parameter p the values for α have been tested in [24]. In own finite-length simulations of extrinsic Ch-Py decoding of extended Hamming product codes it turned out that the best performance in the waterfall region is achieved with $\alpha = 0.5$ and not with $\alpha = 0.4375$, although we have to admit that the differences are almost negligible and the extended Hamming product codes are very "robust" concerning scaling anyways. Therefore we simply stick to $\alpha = 0.5$ in the following definition.

Algorithm 5

Extrinsic Chase-Pyndiah Decoding with Extrinsic Information Scaling by $\boldsymbol{\alpha = 0.5}$

- Chase decoder with p LRBPs and BDD decoding of test list.
- Bit-wise extrinsic Chase decoder according to (5.3).
- In case of an empty competing list in the Chase decoder $\beta = 1$.
- Extrinsic information scaling by $\alpha = 0.5$.
- Final binary decision based on $\mathrm{L}^{\mathrm{APP}}$ (4.25).

We further make the interesting observation concerning Algorithm 5 that there is a tremendous reduction of cases, where an empty competing list in the Chase decoder is encountered and extrinsic information values are computed according to $\mathrm{L}_i^{\mathrm{E}} = \beta d_i$.

Code	t	Algorithm	p	$E_b/N_0 = 1$ dB	$E_b/N_0 = 3$ dB	$E_b/N_0 = 5$ dB
(32,26,4)	1	Alg. 5 (extr.)	4	0%	0%	0%
		Alg. 1 (intr.)	4	46.7%	52.1%	54.2%
		Alg. 5 (extr.)	5	0%	0%	0%
		Alg. 1 (intr.)	5	22.0%	25.1%	25.9%
(64,57,4)	1	Alg. 5 (extr.)	4	0%	0%	0%
		Alg. 1 (intr.)	4	70.1%	72.9%	75.6%
		Alg. 5 (extr.)	5	0%	0%	0%
		Alg. 1 (intr.)	5	51.1%	53.8%	56.1%
(32,21,6)	2	Alg. 5 (extr.)	4	1.3%	4.8%	7.7%
		Alg. 1 (intr.)	4	54.5%	66.1%	72.3%
		Alg. 5 (extr.)	5	0.002%	0.01%	0.03%
		Alg. 1 (intr.)	5	31.6%	42.4%	47.1%
(64,51,6)	2	Alg. 5 (extr.)	4	1.3%	3.0%	6.6%
		Alg. 1 (intr.)	4	71.8%	77.7%	84.3%
		Alg. 5 (extr.)	5	0.003%	0.04%	0.09%
		Alg. 1 (intr.)	5	54.2%	61.4%	68.1%

Table 5.1.: Statistics on the frequency of an empty competing list in the extrinsic and intrinsic Chase decoding process.

The reason for this is the fact, that $\mathrm{L}_i^{\mathrm{IN}} = 0$, when $\mathrm{L}_i^{\mathrm{E}}$ is computed. Therefore, bit position i will always be among the LRBPs and will be flipped for the test list generation, which increases the chance that a competing codeword exists.

In Table 5.1 are given statistics on how often the Chase decoder runs into the empty competing list case for standard intrinsic Ch-Py (Alg. 1) and for extrinsic Ch-Py (Alg. 5), respectively. The percentage values are based on simulation results of a large number of packets encoded with a stand-alone block code and transmitted over a bi-AWGN channel at different SNRs. After that a Chase decoder from Algorithm 1 and Algorithm 5 with different values for p is applied.

For extrinsic Chase decoding of extended Hamming codes with $p \geq 4$, we never encounter the case where β has to be used for the extrinsic information calculation. This comes from the special algebraic structure of Hamming codes. Assume the test list is generated by flipping $p \geq 4$ LRBPs and is decoded into a candidate list afterwards. Then this candidate list always will offer a competing codeword for bit position i, which was set to 0 at the input of the extrinsic Chase decoder and is therefore consequently always among the flipped LRBPs.

But for different eBCH codes with error correcting capability $t \geq 2$ there is still a small fraction of β-cases. This has impact on the performance since we have not defined a suit-

able rule for how to choose a β value and thus have simply set $\beta = 1$. In Algorithm 1 the normalization of the mean of all extrinsic information values helps to give β a meaningful weight. Interestingly, for both, extrinsic and intrinsic Chase decoding, the percentage of applying the β-rule grows with the SNR.

The performance of extrinsic Ch-Py decoding based on Algorithm 5 compared to standard intrinsic Ch-Py from Algorithm 1 for a product code with (32,26,4) Hamming component codes is shown in Figure 5.2. We compare the FER at $I = 8$ decoding iterations and $p = 4$ as the number of LRBPs for the test list creation.

Additionally, iterative decoding of the product code with optimal MAP component decoders and $I = 20$ decoding iterations is plotted. The results are taken from [23], where the BCJR algorithm was used for calculating the extrinsic information in each iteration. Because of finite length effects of the product code over the iterations, scaling the extrinsic information is crucial, as well. Therefore, BCJR with weighted extrinsic information (w.e.i.) shows a significant performance gain in the error-floor region. Ch-Py from Algorithm 1 and 5 do not show large differences in their performance curves, but the error floor of the intrinsic algorithm version 1 is slightly higher. The observation that the error floor of both Ch-Py decoders is above the one of iterative BCJR (w.e.i.) is caused by the suboptimality of the Chase decoder and the lower number of decoding iterations I, but also confirms the findings in [24].

Whereas the FER in the waterfall region of the product codes is very good, the random coding bound (RCB) gets a long way off in the error floor region. This comes from the fact, that the product code error floor is well approximated by the truncated UB [35], which is a function of the minimum distance multiplicities of the component codes.

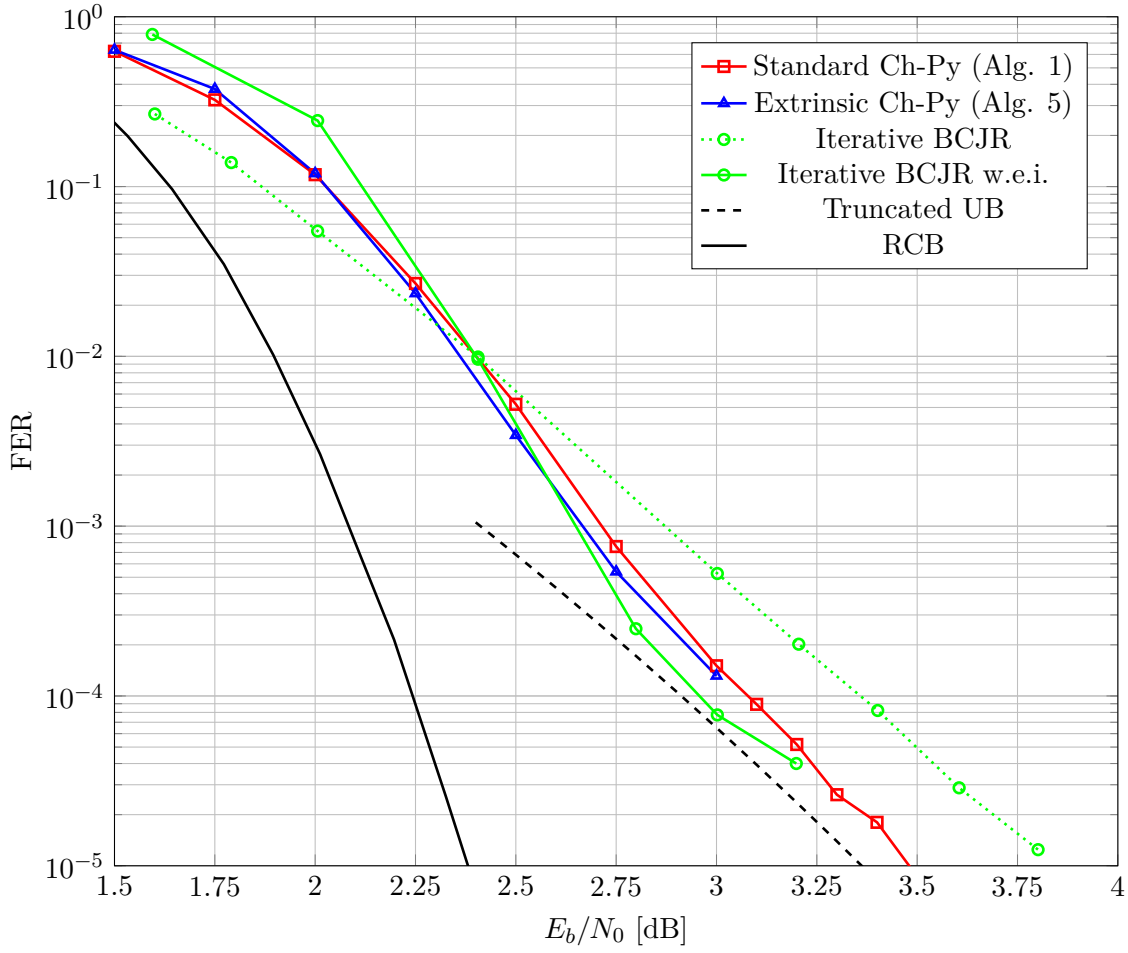


Figure 5.2.: Product Code with (32,26,4) eHAM component code decoded via different decoding algorithms and their gap to truncated UB.

Algorithm	BDD runs
iBDD (IMP)	$n_c + n_r$
EMP	$n_c \cdot 2n_r + n_r \cdot 2n_c$
Standard Ch-Py (Alg. 1)	$n_c \cdot 2^p + n_r \cdot 2^p$
Extrinsic Ch-Py (Alg. 5)	$n_c \cdot n_r \cdot 2^p + n_r \cdot n_c \cdot 2^p$

Table 5.2.: Number of BDD runs per iteration for different product code decodin algorithms.

5.1.3. Computational Complexity

A reasonable method to measure the decoding complexity of a product code decoding algorithm is the number of BDD decoding attempts per iteration. Assume a product code, whose rows are encoded with a binary linear block code of length n_r and columns with a length n_c code, respectively. Then the number of BDD attempts per iteration for the different decoding algorithms presented in this thesis, is given in Table 5.2.
For the HD algorithm iBDD each row and column is decoded only once per iteration. For the extrinsic HD counterpart EMP, each bit in every row and each bit in every column is decoded twice with BDD. For the SD Ch-Py algorithms, a testl ist of size 2^p is decoded with BDD. For intrinsic standard Ch-Py (Alg. 1), the test list is generated row- and columnwise, whereas for the extrinsic algorithm (Alg. 5) this is done bitwise for all rows and columns, which means a huge effort.
However, we do not propose to use Algorithm 5 in practice, but only for asymptotic analysis in the following.

5.2. Extrinsic Chase-Pyndiah Decoding with Postprocessing

Choosing $\alpha = 0.5$ and $\beta = 1$ in Algorithm 5 turns out to be a good choice of parameters, when extended Hamming codes are used as product code component codes. Indeed, for constructions with codes offering a larger error correcting capability t, the scaling for extrinsic Ch-Py decoding from Algorithm 5 has to be refined.
We decide for extrinsic information postprocessing and want to use the same strategy as in Subsection 4.2.1. Note that the functions in Table 4.2 were obtained by the Chase decoder output samples stemming from an intrinsic Chase decoder. But since we deal with a different algorithm in this chapter, we have to generate new samples of $\mathrm{L}^{\mathrm{E,Chase}}$ to get the valid function f_{pp} of the form (4.12) for this extrinsic case, too. The corresponding scaling factors γ and δ for the tanh function in (4.12) are therefore given in Table 5.3.

Code	p	γ	δ
(16,7,6)	4	$2.7714 \cdot \exp(2.7388 \cdot \mathrm{MI})$	$-0.1585 \cdot \mathrm{MI} + 0.1633$
(16,7,6)	5	$1.7946 \cdot \exp(3.3353 \cdot \mathrm{MI})$	$-0.2316 \cdot \mathrm{MI} + 0.2269$
(32,21,6)	4	$0.3298 \cdot \exp(4.7247 \cdot \mathrm{MI})$	$-0.1136 \cdot \mathrm{MI} + 0.1494$
(32,21,6)	5	$0.3688 \cdot \exp(4.7989 \cdot \mathrm{MI})$	$-0.2092 \cdot \mathrm{MI} + 0.2260$
(64,51,6)	4	$0.0051 \cdot \exp(8.8890 \cdot \mathrm{MI})$	$-0.0581 \cdot \mathrm{MI} + 0.0998$
(64,51,6)	5	$0.0127 \cdot \exp(7.9880 \cdot \mathrm{MI})$	$-0.2297 \cdot \mathrm{MI} + 0.2637$
(128,113,6)	4	$0.0019 \cdot \exp(9.6701 \cdot \mathrm{MI})$	$-1.1365 \cdot \mathrm{MI} + 1.2599$
(128,113,6)	5	$0.0024 \cdot \exp(9.1227 \cdot \mathrm{MI})$	$-1.3404 \cdot \mathrm{MI} + 1.4081$

Table 5.3.: Posprocessing functions for extrinsic Chase-Pyndiah decoding.

When combining the postprocessing functions from Table 5.3, the Ch-Py algorithm with postprocessing based on estimated mutual information (Alg. 3) and the extrinsic Ch-Py version (Alg. 5), we can formalize Algorithm 6 as follows.

ALGORITHM 6

Extrinsic Chase-Pyndiah Decoding with Extrinsic Information Postprocessing Based on Estimated Mutual Information Values

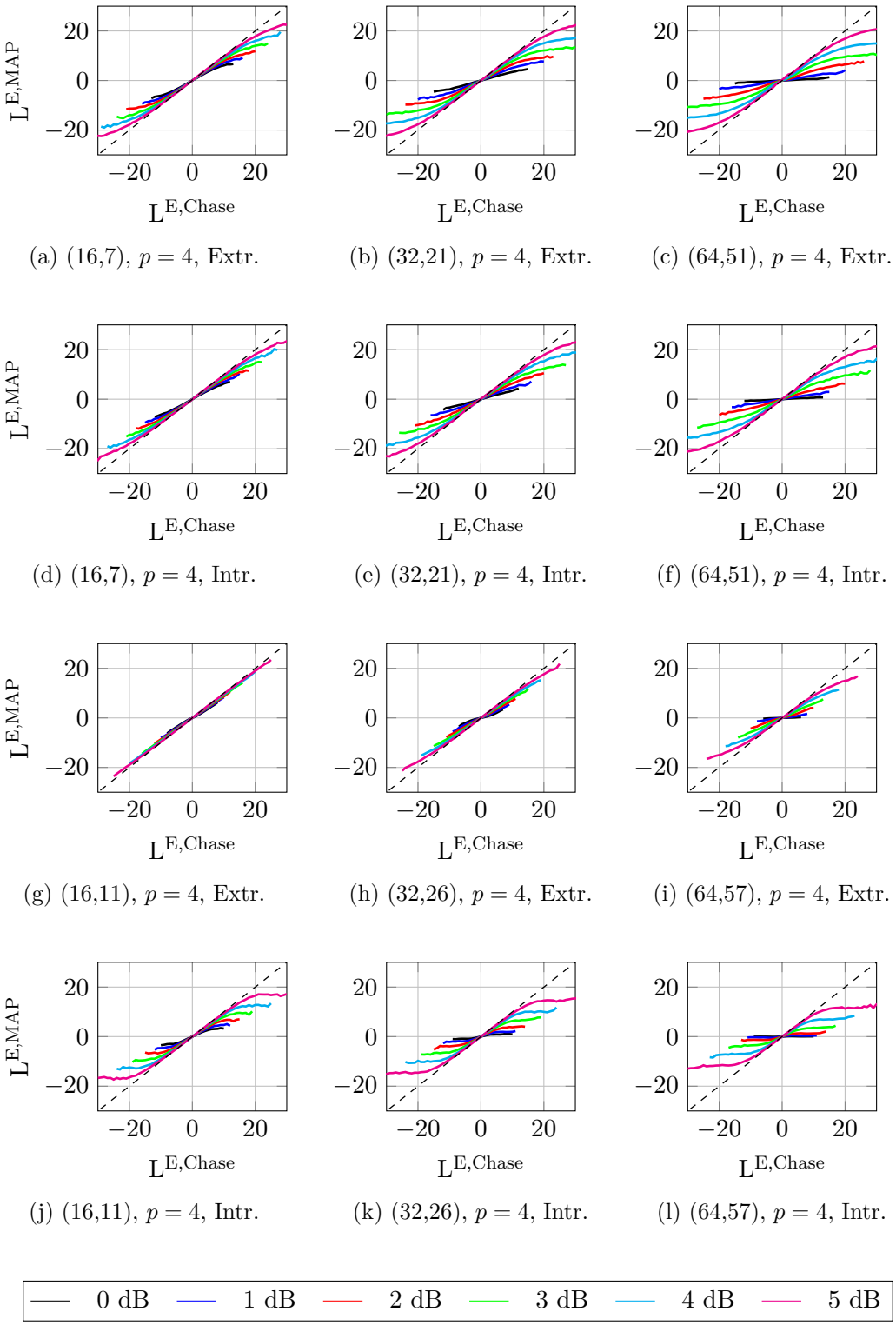
- Chase decoder with p LRBPs and BDD decoding of test list.
- Bit-wise extrinsic Chase decoder according to (5.3).
- Instantaneous estimation of $\mathrm{I}\left(\mathrm{L}^{\mathrm{IN}}; X\right)$ by (4.17).
- Postprocessing of $\mathrm{L}^{\mathrm{E},\alpha}$ according to (4.23) based on f_{pp} from Table 5.3.
- Postprocessing of $\mathrm{L}^{\mathrm{E},\beta}$ according to (4.24).
- Final binary decision based on $\mathrm{L}^{\mathrm{APP}}$ (4.25).

In Table 5.3 are only given γ and δ for eBCH codes with $t = 2$. For extended Hamming codes one could observe that the mean curves are very close to the $\mathrm{L}^{\mathrm{E,Chase}} = \mathrm{L}^{\mathrm{E,MAP}}$ line. This means that extrinsic Chase decoding almost produces the optimal MAP extrinsic output and therefore no postprocessing is necessary.
Of course it would be beneficial to derive postprocessing functions for larger blocklengths too, but the generation of MAP extrinsic information samples based on HR decoding according to (4.2) becomes infeasible for larger redundancy $n - k$. In order to draw

conclusions concerning the shape of f_{pp} for larger blocklengths n, we want to investigate if there are trends for increasing n.
In Figure 5.3 are plotted interpolated sample curves at different E_b/N_0 levels as already shown in Figure 4.4. However, we fixed $p = 4$ and arranged the curves in subfigures for six different codes for intrinsic and extrinsic Chase decoding. We want to analyze in subfigures (a) - (f) eBCH codes with $t = 2$ and in subfigures (g) - (l) extended Hamming codes with $t = 1$.
First of all, we can see that the overestimation of $\mathrm{L}^{\mathrm{E,Chase}}$ is introduced by both decoders for all codelengths, because especially for increasing magnitudes of extrinsic information the absolute mean value of the MAP output is smaller than the Chase output, i.e.,

$$|\mathrm{L}^{\mathrm{E,MAP}}| < |\mathrm{L}^{\mathrm{E,Chase}}|. \tag{5.4}$$

Moreover, for increasing blocklength n this overestimation becomes larger, which can be observed by the growing deviation between the curves and the black dashed line, where $\mathrm{L}^{\mathrm{E,Chase}} = \mathrm{L}^{\mathrm{E,MAP}}$.
Secondly, when comparing extrinsic and intrinsic Chase decoding there is not a big difference in the curves for eBCH codes with $t = 2$. However, for the extended Hamming codes, we observe excellent outputs by the extrinsic Chase decoder, because they are almost equal to the optimal MAP outputs. This is also the reason why there are no postprocessing functions for extended Hamming codes provided in Table 5.3 for Algorithm 6. Nevertheless, the $\mathrm{L}^{\mathrm{E,Chase}}$ values for the (64,57,4) code start to show a deviation from the black dashed line. One may assume that this trend is continued for increasing n. This intuition is also confirmed when looking at simulations of product codes with larger extended Hamming component codes. The performance gets better when a constant attenuation of $\alpha = 0.5$ is used for all iterations. The corresponding postprocessing function would therefore be just a linear function with slope 0.5, regardless the SNR.
The interpolated sample curves for intrinsic Chase decoding for extended Hamming codes show a quite different shape, though. It is remarkable that for various E_b/N_0, the curves very strictly saturate at a certain level. In other words, when we apply intrinsic Chase decoding to an extended Hamming code, the absolute average value of the generated extrinsic output $\mathrm{L}^{\mathrm{E,Chase}}$ will not exceed a certain bound. This behavior is visible for all simulated codelengths of extended Hamming codes and is therefore very likely to appear for larger n as well.



(a) (16,7), $p = 4$, Extr. (b) (32,21), $p = 4$, Extr. (c) (64,51), $p = 4$, Extr.

(d) (16,7), $p = 4$, Intr. (e) (32,21), $p = 4$, Intr. (f) (64,51), $p = 4$, Intr.

(g) (16,11), $p = 4$, Extr. (h) (32,26), $p = 4$, Extr. (i) (64,57), $p = 4$, Extr.

(j) (16,11), $p = 4$, Intr. (k) (32,26), $p = 4$, Intr. (l) (64,57), $p = 4$, Intr.

Figure 5.3.: Development of postprocessing functions for the extrinsic and intrinsic version of Chase decoding with $p = 4$ for different codes.

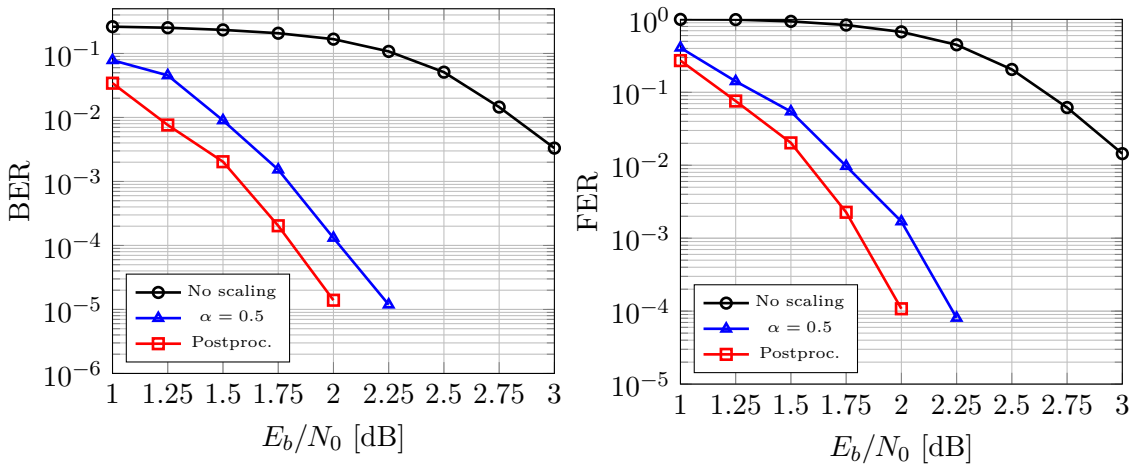


Figure 5.4.: Simulation results of a $(32, 21, 6)^2$ product code decoded with 8 iterations of extrinsic Chase-Pyndiah decoding ($p = 5$) with $\alpha = 0.5$ compared to extrinsic information postprocessing.

We want to apply Algorithm 6 in an iterative product decoding simulation with a $t = 2$ error correcting component code to determine the expected gain by postprocessing. According to Table 5.1 there is only a small negligible fraction of β values in extrinsic Chase decoding for a choice of $p \geq 4$. The advantage of the extrinsic Ch-Py version therefore simplifies postprocessing, because there are almost no cases where $\mathrm{L}^{\mathrm{E,Chase}} = \pm\beta$, so that the impact of β on the BER statistics becomes very small.

For the following simulation, we will set $p = 5$ LRBPs and simulate $I = 8$ iterations of a product code with (32,21,6) eBCH component codes. Figure 5.4 shows the performances of extrinsic Ch-Py with not scaling L^{E} in Algorithm 5 at all (black), attenuating L^{E} constantly with $\alpha = 0.5$ (blue - Alg. 5) and postprocessing L^{E} (red - Alg. 6).

First of all one can observe that no scaling of extrinsic outputs results in a massive error propagation. As already shown in the scatterplot in Figure 4.3, the Chase decoder highly overestimates its extrinsic output and therefore the decoder only can hardly recover from wrong reliabilities. Because of the attenuation with $\alpha = 0.5$ the same BER can be achieved with a SNR, which is several magnitudes smaller compared to no scaling.

The constant scaling factor $\alpha = 0.5$ is outperformed by employing the postprocessing function from Table 5.3. We can observe an additional gain of approximately 0.2 dB in terms of BER and FER, which is a result of the theoretically grounded attenuation by f_{pp} dependent on the estimated MI.

5.3. Density Evolution

In this section, we derive a DE analysis for product codes, decoded with extrinsic Ch-Py decoding as in Algorithm 5 and 6. For less complex decoding algorithms of product codes, such as iBDD and iBDD-SR there is already rigorous analysis on the asymptotic decoding behavior of the corresponding GLDPC ensemble [38, 46]. For these cases, DE follows an analytical approach based on the Gaussian distribution of the channel LLR under the all-zero codeword assumption. However, the messages exchanged in iBDD and iBDD-SR are HD messages and the CN update is easier to analyze compared to Ch-Py. In [60] the error rate performance of a Chase decoder was derived analytically. Based on those results, Lehmann and Maggio tried to obtain iterative decoding thresholds of product codes decoded via Ch-Py by using the Gaussian approximation [61]. The results may be treated with caution, since the authors did neither consider the intrinsic nature of standard Ch-Py, nor the use of scaling factors.
We propose an alternative method for DE based on a Monte-Carlo (MC) method in this section and compare the derived iterative decoding thresholds to the ones in [61] and to the Shannon limit, respectively.

5.3.1. A Density Evolution Approach based on a Monte-Carlo Method

The underlying idea of the approach is formulated in [62] and will be introduced accordingly in the following. To mitigate finite-length effects, an "infinitely" long GLDPC code is simulated. This can be done by choosing a very large number N of VNs and

$$M = \frac{2N}{n} \tag{5.5}$$

CNs, where n is the length of the component code.
The VNs are initialized with a channel LLR, distributed according to the all-zero codeword assumption, i.e.,

$$\mathrm{L}^{\mathrm{CH}} \sim \mathcal{N}(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}) \tag{5.6}$$

and passed over a random edge permutation $\pi^{(1)}$ to the CNs, where the extrinsic information is computed according to the extrinsic Ch-Py update rule. It is worth pointing out, that in this context the E_b/N_0 for the corresponding channel noise variance σ^2 has to be computed with respect to the rate of a GLDPC code given in (3.6).
As defined in Section 5.1, extrinsic Ch-Py decoding means that every CN performs n times the Chase algorithm, to generate $\mathrm{L}_i^{\mathrm{E}}$ for the i-th of n incoming branches, where the decoder input LLR $\mathrm{L}_i^{\mathrm{IN}}$ is set to zero, when $\mathrm{L}_i^{\mathrm{E}}$ is computed. All extrinsic messages $\mathbf{L}^{\mathrm{E}}$

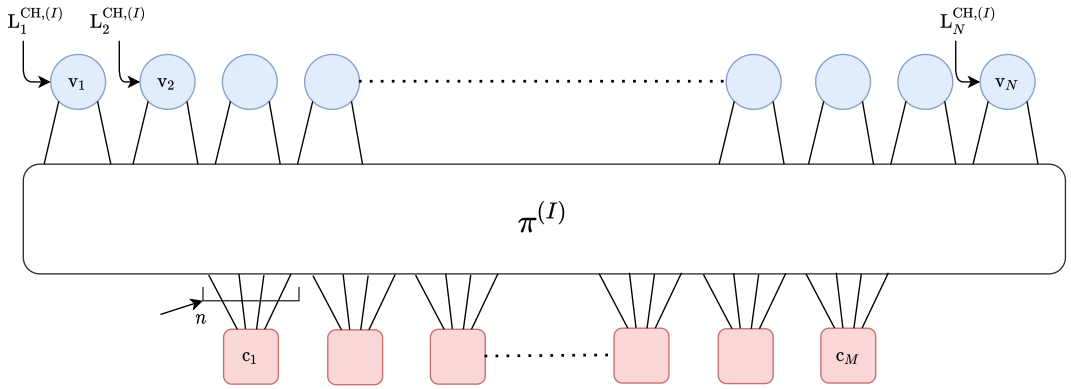


Figure 5.5.: Density Evolution of a GLDPC code ensemble based on a Monte-Carlo method.

are sent back over $\pi^{(1)}$ to the VNs after that. Then the first iteration is finished.
For the second iteration $I = 2$, new channel LLRs are generated and the input for the CNs is computed as

$$\mathbf{L}^{\mathrm{IN}} = \mathbf{L}^{\mathrm{CH}} + \mathbf{L}^{\mathrm{A}}, \tag{5.7}$$

where the a priori information $\mathbf{L}^{\mathrm{A}}$ corresponds to a scaled version of the extrinsic information of the last iteration, i.e.,

$$\mathbf{L}^{\mathrm{A},(I)} = \alpha \mathbf{L}^{\mathrm{E},(I-1)}. \tag{5.8}$$

This input LLR is sent over a new random permutation $\pi^{(2)}$, which mimics the behavior of an infinitely long code [62]. The setup for each iteration of this MC method is additionally visualized in Figure 5.5.
The procedure is continued until a maximum number of iterations $I_{\max}$ is reached or until decoding has converged. Convergence is achieved when the negative area below the PDF of the extrinsic information LLRs distribution, which has approximately the shape of Gaussian curve, goes to zero when I grows to ∞. This area corresponds to the error probability and will be denoted by ϵ. For a finite, but very large number of VNs, we may compute ϵ simply by the fraction of negative values in the $\mathbf{L}^{\mathrm{E}}$ matrix over the number of VNs.
To obtain an approximately Gaussian distribution, the value of LRBP p has to be adjusted to ensure a Chase decoder test list, which is large enough to guarantee the existence of a competing codeword. If there is no competing codeword in the candidate list $\mathcal{L}$, the extrinsic output is obtained by (3.18). Too many of these extrinsic information values would result in two peaks in the distribution at $\pm\beta$ as in Figure 4.7. In Figure 5.6 the

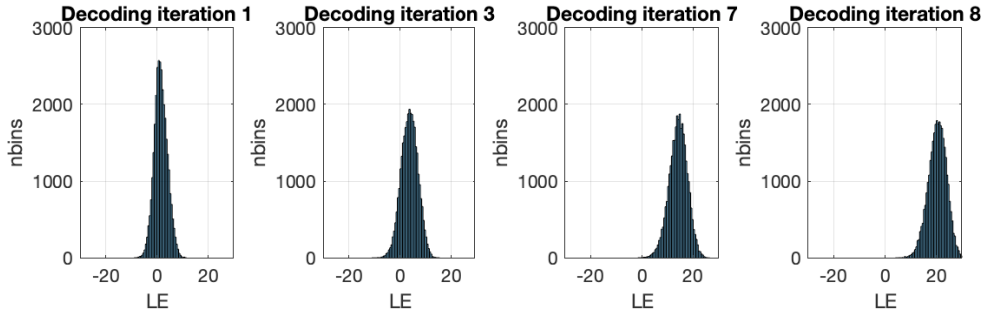


Figure 5.6.: Evolution of the extrinsic information distribution over the decoding iterations .

histograms approximating the PDFs, are plotted for different decoding iterations. For this example, a (32,26,4) extended Hamming code and extrinsic Chase decoding with $p = 4$ was used at the CNs. The choice of p turns out to be sufficient to avoid the empty competing list case according to Table 5.1. In this example, a SNR above threshold was used, because decoding has already converged in iteration 8. Since we do not impose a constraint on the number of iterations, decoding may also converge with a lower SNR at the expense of more iterations. Based on error probability ϵ, the iterative decoding threshold of the underlying GLDPC code ensemble can be found. This threshold is defined as

$$(E_b/N_0)^\star = \inf\left\{E_b/N_0 : \lim_{I\to\infty} \epsilon = 0\right\}, \tag{5.9}$$

,i.e., the infimum of all E_b/N_0 for which ϵ becomes zero when the number of iterations I goes to infinity. An efficient method for finding $(E_b/N_0)^\star$ with high speed and good accuracy is the bisection method [63, Appendix. A.6.].

Component Code	(32,26,4)	(64,57,4)	(128,120,4)
Product Code Rate R_{PC}	0.6602	0.7932	0.8789
GLDPC Code Rate R_{GLDPC}	0.6250	0.7812	0.8750
$(E_b/N_0)^\star$ from MC DE (Alg. 5 - $\alpha = 0.5$)	1.70 dB	2.61 dB	3.62 dB
$(E_b/N_0)^\star$ from MC DE (Alg. 5 - $\alpha =$(3.21))	1.86 dB	2.74 dB	3.70 dB
$(E_b/N_0)^\star$ from [61]	0.7 dB	2.2 dB	3.3 dB
Shannon SD Limit	1.02 dB	1.97 dB	2.89 dB

Table 5.4.: DE thresholds of the MacKay Monte-Carlo approach for product codes with extended Hamming component codes ($t = 1$) decoded via extrinsic Chase-Pyndiah decoding with $p = 4$.

5.3.2. Iterative Decoding Thresholds for Chase-Pyndiah Decoding

Extended Hamming Component Codes

Finally we want to compute the iterative decoding thresholds for different GLDPC code ensembles with the MC DE method from the previous subsection. We start with the extended Hamming component codes ($t = 1$) and can therefore scale the extrinsic information for the next iteration simply with a constant scaling factor $\alpha = 0.5$. According to Table 5.1 we do not run into the case of an empty competing list when $p = 4$.
To get reliable thresholds, 10.000 CNs and a maximum of 50 iterations is used. The extrinsic Ch-Py decoder based on Algorithm 5 is actually defined by a scaling of $\alpha = 0.5$, but for a further comparison the performance of using Pyndiah's iteration-dependent scaling factors in (3.21) is investigated, too. Finally, we want to compare the obtained $(E_b/N_0)^\star$ to the ones in [61], where Lehmann and Maggio used an analytical approach. The results are given in Table 5.4.

In general, the choice of $\alpha = 0.5$ is to be preferred over Pyndiah's scaling factors (3.21), although the difference between the respective thresholds shrinks for larger blocklengths n. We can also observe that for increasing n, the gap between the MC thresholds and the ones from the analytical method in [61] becomes smaller, too. But obviously the analytical results are not very reliable, especially for product codes with small component code blocklengths, where they even lie beyond the Shannon limit. Interestingly, for all calculated MC thresholds with $\alpha = 0.5$, the gap to capacity is approximately only 0.7 dB, regardless the coderate.
In Figure 5.7 one can see the performance of finite-length simulations with $I = 8$ decoding iterations of Algorithm 5 applied to product codes with extended Hamming component

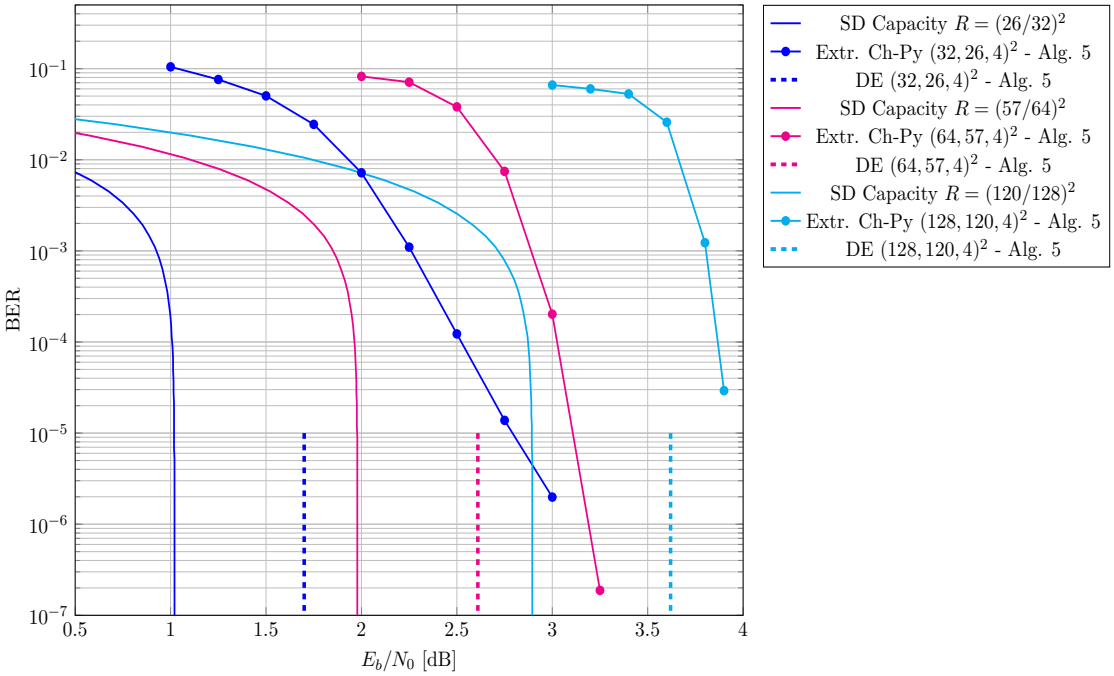


Figure 5.7.: Simulation of different length product codes with $t = 1$ component codes decoded with extrinsic Chase decoding ($p = 4$) at the CNs and their corresponding ensemble DE thresholds.

codes. For the simulations and the computation of the corresponding GLDPC ensemble thresholds, $\alpha = 0.5$ and $p = 4$ LRBPs was used. It becomes clear, that the thresholds yield a better prediction on the BER for larger blocklenghts, but in general they resemble very well the beginning of the waterfall region for the simulated code.

Component Code	(32,21,6)	(64,51,6)	(128,113,6)
Product Code Rate R_{PC}	0.4307	0.6350	0.7794
GLDPC Code Rate R_{GLDPC}	0.3125	0.5938	0.7656
$(E_b/N_0)^\star$ from MC DE (Alg. 5 - $\alpha = 0.5$)	1.29 dB	2.14 dB	2.94 dB
$(E_b/N_0)^\star$ from MC DE (Alg. 5 - α =(3.21))	1.23 dB	2.05 dB	2.87 dB
$(E_b/N_0)^\star$ from MC DE (Alg. 6 - Postproc.)	1.11 dB	1.89 dB	2.72 dB
$(E_b/N_0)^\star$ from [61]	-0.2 dB	1.5 dB	2.7 dB
Shannon SD Limit	-0.11 dB	0.87 dB	1.86 dB

Table 5.5.: DE thresholds of the MacKay Monte-Carlo approach for product codes with extended BCH component codes ($t = 2$) decoded via extrinsic Chase-Pyndiah decoding with $p = 5$.

Extended BCH Component Codes

DE analysis and extrinsic Ch-Py decoding based on Algorithm 5 with $\alpha = 0.5$ for eBCH component codes with higher error correcting capability ($t \geq 2$) is suboptimal. The observation that eBCH codes with larger t are more sensitive concerning error propagation in the iterative process is mainly based on two reasons:

- Extrinsic Chase-Pyndiah decoding based on the GLDPC principle still runs into the case where there is no competing codeword and $\mathrm{L}^{\mathrm{E,Chase}}$ has to be computed according to (5.2).
- For eBCH component codes with $t \geq 2$ postprocessing according to Algorithm 6 may be preferred over a constant scaling.

To overcome the first issue, we decide to run DE with $p = 5$. When looking at the statistics in Table 5.1 we can observe that for $p = 5$ LRBP used for the test list generation of the Chase decoder, the fraction of β values becomes vanishingly small. The extrinsic information scaling problem can easily be solved by using the postprocessing functions from Table 5.3 as explained in Algorithm 6.
In Table 5.5 are given the thresholds $(E_b/N_0)^\star$ for the GLDPC ensembles consisting of eBCH codes with $t = 2$. and the reference thresholds from Lehmann and Maggio's analytical method [61]. As a further comparison, the suboptimal DE threshold of our MC method with $\alpha = 0.5$, α from (3.21) and the Shannon limit are provided.
The thresholds of the MacKay MC method with postprocessing show a significant gain compared to the ones obtained with a constant scaling of $\alpha = 0.5$. This was already visible in the simulation results in Figure 5.4 as well. We can also observe that the $(E_b/N_0)^\star$

values from [61] are much lower, but since the value for the (32,21,6) eBCH component code is again even below the Shannon limit, we cannot be sure if they can be seen as a valid reference here. At least for small codelengths, finite-length effects in the analytical derivation by Lehmann and Maggio seem to have a fatal influence on the threshold calculation.

A further aspect, which must be considered when interpreting the results in Table 5.4 and 5.5 is the fact that the rate of a GLDPC code as defined in (3.6) was considered for the calculation of $(E_b/N_0)^\star$. Since the formula for the rate of a product code is given by $R_{\mathrm{PC}} = k^2/n^2$, we can see that for small component code rates, the rate of the GLDPC code $R_{\mathrm{GLDPC}} = 2 \cdot k/n - 1$ is smaller than the one of the product code. Although, for increasing component code rate, the GLDPC rate converges to the one of the product code. This rate deviation has impact on the E_b/N_0 calculation in (2.22), where the rate R is in the denominator. Consequently, since $R_{\mathrm{GLDPC}} < R_{\mathrm{PC}}$ we may expect the calculated threshold $(E_b/N_0)^\star$ for the GLDPC ensemble to be too large. But surprisingly this is not the case, because the simulated "infinitely" long GLDPC code in the MC DE approach with random edge permutation, shows a faster convergence behavior as the product code. In other words, although the rate R_{PC} of the product code is larger than the one of its GLDPC ensemble with rate R_{GLDPC}, the threshold $(E_b/N_0)^\star$ of the ensemble is indeed matching for the product code, because the rate deviation is compensated by the difference in terms of SNR, for which the GLDPC ensemble converges earlier.

Analogous to the extended Hamming component codes, we plot in Figure 5.8 the BER results of decoding product codes from their corresponding GLDPC ensemble with $t = 2$ eBCH component codes based on Algorithm 6. In the simulations we used $p = 5$ LRBP and $I = 8$ iterations. As reference, the BER of extrinsic Ch-Py decoding with $\alpha = 0.5$ (Alg. 5) is plotted. The iterative decoding threshold of the GLDPC ensemble for both scaling methods and the Shannon limit for the corresponding product code rate are provided as well.

In general, similar conclusions can be drawn from the results in Figure 5.8 like for the $t = 1$ component codes before. The DE approach based on the MC method again delivers thresholds, which match very well the beginn of the waterfall region of the simulated GLDPC codes. We can also observe that the gap to SD capacity for the $t = 2$ eBCH component code thresholds is larger than for the ones in Figure 5.7, where $t = 1$ component codes were used. For the $(64, 51, 6)$ GLDPC ensemble the Shannon limit is 1 dB away and for $(32, 21, 6)$ even 1.2 dB. But we may remark here that the aim in this thesis is to analyze the performance for codes with high rates and therefore the product codes in Figure 5.8 are not in our main scope and other code classes may yield better performance with comparable complexity.

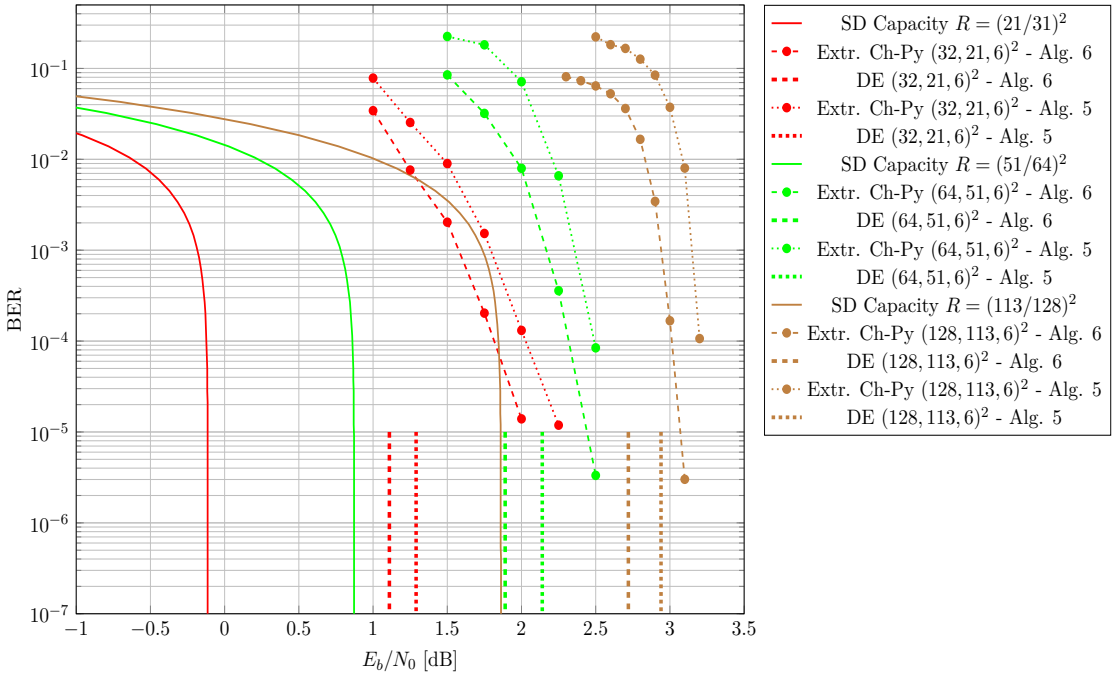


Figure 5.8.: Simulation of different length product codes with $t = 2$ component codes decoded with extrinsic Chase decoding ($p = 5$) at the CNs and their corresponding ensemble DE thresholds.

When comparing both methods of extrinsic information scaling, i.e., $\alpha = 0.5$ and postprocessing, it becomes visible that the gap between the simulated GLDPC codes corresponds to the same difference between the MC DE thresholds of their corresponding ensemble. In other words, the gains of certain choices of extrinsic information scaling in the asymptotic analysis of the "infinitely" long GLDPC codes can be directly translated into the gain of finite-length simulations. Since those can be very time consuming, the threshold calculation delivers an appealing alternative for the evaluation of α or different scaling approaches.

6. Conclusion

New strategies to improve the BER performance of product codes, decoded via the Chase-Pyndiah algorithm have been proposed in this thesis. The first approach follows an intuition, that the convergence state in the iterative process may be measured by the fraction of valid component codewords in the product code array. By leaving Pyndiah's decoder setup untouched and replacing the predefined scaling coefficient by this described fraction, a considerable gain has been made possible. The gain is the result of an instantaneous weighting, which is able to react on divergence behavior by stronger attenuation of the extrinsic messages in order to avoid error propagation over the decoding iterations. Especially for product codes with error correcting capability $t = 2$, this approach proved as highly advisable for practical implementations, where the syndromes are computed anyways in general and therefore this method comes with no increase in complexity.
The other strategy to enhance the scaling of the Chase decoder extrinsic output is based on the finding that not Pyndiah's coefficients, but the combination with normalization of the mean absolute value of this output to one, is the key tool for achieving such good BERs with very low complexity. We have shown that the Chase decoder highly overestimates its extrinsic output due to its suboptimal nature. Postprocessing functions for different codelengths have been derived. They can be applied in the iterative process to attenuate the extrinsic Chase decoder output, so that it approximates the magnitude of the corresponding MAP value. A performance gain has also been achieved with this postprocessing method in finite-length simulations.

Furthermore, asymptotic decoding analysis of the Ch-Py algorithm has been studied in this thesis. Because of the deterministic product code structure and the intrinsic nature of Chase decoding, DE analysis is inconclusive. However, product codes are an instance of GLDPC codes and therefore DE can be applied to the GLDPC ensemble, which the corresponding product code belongs to. We have introduced a modified version of Ch-Py decoding ensuring the exchange of fully extrinsic messages. The new algorithm is applied at the CNs of an infinitely long GLDPC code [62]. DE can be implemented by a MC simulation of a very large GLDPC code with a random edge permutation in every iteration. The thresholds of different component codelengths have been computed and have

been compared to a different DE approach for Ch-Py decoding.

In further research, the iterative decoding thresholds of product codes with MAP algorithms at the CNs may be computed with the MC DE approach. These optimal MAP thresholds could be compared to their Ch-Py equivalent counterpart, in order to estimate the SNR gap between the algorithms in the asymptotic limit of large block length.
Another interesting task for researchers would be to derive postprocessing functions for larger BCH codes, too. Since the computation of MAP extrinsic information values for increasing code redundancy becomes infeasible, the generation of extrinsic information samples is a computationally expensive effort. However, the question if and how much performance gain by postprocessing can be achieved for larger product code constructions is unclear.
Since every extrinsic value is given into the postprocessing function, the tanh function chosen in this thesis, is too complex for practical systems, an easier approximation may be found here.

A. Acronyms

AD anchor decoding

APP a-posteriori probability

ASK amplitude shift keying

AWGN additive white Gaussian noise

BCH Bose-Chaudhuri-Hocquenghem

BCJR Bahl-Cocke-Jelinek-Raviv

BDD bounded distance decoding

BEC binary erasure channel

BEEC binary error and erasure channel

BER bit error rate

bi-AWGN binary input additive white Gaussian noise

BM Berlekamp-Massey

BMP-GMDD binary message passing decoding based on GMDD

BP belief propagation

bpcu bits per channel use

BPSK binary phase shift keying

BSC binary symmetric channel

CN check node

Ch-Py Chase-Pyndiah

DE density evolution

eBCH extended BCH

EMP extrinsic message passing

ETPD extrinsic turbo product decoding

EXIT extrinsic information transfer

FEC forward error correction

FER frame error rate

GLDPC generalized low density parity check

GMDD generalized minimum distance decoding

HD hard decision

HR Hartmann-Rudolph

iBDD iterative bounded distance decoding

iBDD-CR iterative bounded distance decoding with combined reliability

iBDD-SR iterative bounded distance decoding with scaled reliability

iGMDD-SR iterative generalized minimum distance decoding with scaled reliability

IMP intrinsic message passing

LDPC low density parity check

LLR log-likelihood ratio

LRBP least reliable bit positions

MAP maximum-a-posteriori

MC Monte-Carlo

ML maximum likelihood

PAS probabilistic amplitude shaping

PDF probability density function

PSK phase shift keying

QAM quadrature amplitude modulation

RCB random coding bound

SISO soft-input soft-output

SD soft decision

SNR signal to noise ratio

SPC single-parity check

TPD turbo product decoding

UB union bound

VN variable node

WE weight enumerator

WIMAX Worldwide Interoperability for Microwave Access

Bibliography

[1] A. Sheikh, "On hard-decision forward error correction with application to high-throughput fiber-optic communications," Doctoral thesis, Chalmers University of Technology, Gothenburg, Sweden, 2019.

[2] W. D. Grover, "Forward error correction in dispersion-limited lightwave systems," *Journal of Lightwave Technology*, vol. 6, no. 5, pp. 643–654, 1988.

[3] A. Leven and L. Schmalen, "Status and recent advances on forward error correction technologies for lightwave systems," *Journal of Lightwave Technology*, vol. 32, no. 16, pp. 2735–2750, 2014.

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

[5] R. M. Pyndiah, "Near-optimum decoding of product codes: block turbo codes," *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, 1998.

[6] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[7] G. Kramer, "Lecture notes for information theory," *TU Muenchen*, 2019.

[8] G. Böcherer, *Principles of Coded Modulation*. Technische Universität München, 2018. [Online]. Available: https://books.google.de/books?id=rvF8tQEACAAJ

[9] W. Ryan and S. Lin, "Channel codes classical and modern," *Channel Codes: Classical and Modern*, 01 2009.

[10] J. Proakis and M. Salehi, *Digital Communications*. McGraw-Hill, 2008. [Online]. Available: https://books.google.de/books?id=ABSmAQAACAAJ

[11] A. Wachter-Zeh, "Lecture notes for channel coding," *TU Muenchen*, 2019.

[12] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

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

[14] C. Hartmann and L. Rudolph, "An optimum symbol-by-symbol decoding rule for linear codes," *IEEE Transactions on Information Theory*, vol. 22, no. 5, pp. 514–517, 1976.

[15] L. C. Perez, J. Seghers, and D. J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1698–1709, 1996.

[16] R. Roth, *Intoduction to Coding Theory*, 01 2006.

[17] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, no. 1, pp. 68 – 79, 1960.

[18] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, no. 2, pp. 147–56, 1959.

[19] G. Clark and J. Cain, *Error- Correction Coding for Digital Communications*, ser. (Applications of communications theory). Plenum Press, 1981. [Online]. Available: https://link.springer.com/content/pdf/bbm%3A978-1-4899-2174-1%2F1.pdf

[20] V. Pless, *Introduction to the Theory of Error-correcting Codes*, ser. A Wiley-Interscience publication. Wiley, 1989. [Online]. Available: https://books.google.de/books?id=LSvvAAAAMAAJ

[21] J. Massey, "Shift-register synthesis and bch decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.

[22] R. Kotter, "A fast parallel implementation of a berlekamp-massey algorithm for algebraic-geometric codes," *IEEE Transactions on Information Theory*, vol. 44, no. 4, pp. 1353–1368, 1998.

[23] M. Lentmaier, G. Liva, E. Paolini, and G. Fettweis, "From product codes to structured generalized ldpc codes," in *2010 5th International ICST Conference on Communications and Networking in China*, 2010, pp. 1–8.

[24] F. Chiaraluce and R. Garello, "Extended hamming product codes analytical performance evaluation for low error rate applications," *IEEE Transactions on Wireless Communications*, vol. 3, no. 6, pp. 2353–2361, 2004.

[25] V. M. Sidel'nikov, "Weight spectrum of binary bose–chaudhuri–hoquinghem codes," *Problemy Peredachi Informatsii*, vol. 7, no. 1, pp. 14–22, 1971.

[26] R. Morelos-Zaragoza, *Appendix A: Weight Distributions of Extended BCH Codes*, 04 2002, pp. 205–216.

[27] M. Terada. (2018) Weight distribution of primitive and extended bch codes. [Online]. Available: https://isec.ec.okayama-u.ac.jp/home/kusaka/wd/index.html

[28] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[29] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC '93 - IEEE International Conference on Communications*, vol. 2, 1993, pp. 1064–1070 vol.2.

[30] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[31] R. D. Cideciyan, S. Furrer, and M. A. Lantz, "Product codes for data storage on magnetic tape," *IEEE Transactions on Magnetics*, vol. 53, no. 2, pp. 1–10, 2017.

[32] S. Emmadi, K. R. Narayanan, and H. D. Pfister, "Non- volatile memories workshop," in *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, 2015.

[33] J. Justesen, K. J. Larsen, and L. A. Pedersen, "Error correcting coding for otn," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 70–75, 2010.

[34] M. Wang, "Wimax physical layer: Specifications overview and performance evaluation," in *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, 2011, pp. 10–12.

[35] G. Liva, E. Ben Yacoub, B. Matuz, and F. Steiner, "Lecture notes to channel codes for iterative decoding," *TU Muenchen*, 2020.

[36] L. M. G. M. Tolhuizen, "More results on the weight enumerator of product codes," *IEEE Transactions on Information Theory*, vol. 48, no. 9, pp. 2573–2577, 2002.

[37] M. Lentmaier and K. S. Zigangirov, "Iterative decoding of generalized low-density parity-check codes," in *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No.98CH36252)*, 1998, pp. 149–.

[38] Y. Jian, H. D. Pfister, and K. R. Narayanan, "Approaching capacity at high rates with iterative hard-decision decoding," in *2012 IEEE International Symposium on Information Theory Proceedings*, 2012, pp. 2696–2700.

[39] G. Liva, W. E. Ryan, and M. Chiani, "Quasi-cyclic generalized ldpc codes with low error floors," *IEEE Transactions on Communications*, vol. 56, no. 1, pp. 49–57, 2008.

[40] C. Häger and H. Pfister, "Approaching miscorrection-free performance of product codes with anchor decoding," *IEEE Transactions on Communications*, vol. 66, pp. 2797–2808, 2018.

[41] C. Condo, P. Giard, F. Leduc-Primeau, G. Sarkis, and W. J. Gross, "A 9.52 db ncg fec scheme and 162 b/cycle low-complexity product decoder architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, p. 1420–1431, Apr 2018. [Online]. Available: http://dx.doi.org/10.1109/TCSI.2017.2745902

[42] J. Hagenauer, "The turbo principle-tutorial introduction and state of the art," 1997.

[43] A. Sheikh, A. G. i Amat, and G. Liva, "Iterative bounded distance decoding of product codes with scaled reliability," 2018. [Online]. Available: https://arxiv.org/abs/1805.05270

[44] A. Sheikh, A. G. i Amat, G. Liva, and A. Alvarado, "Refined reliability combining for binary message passing decoding of product codes," 2020.

[45] G. Forney, "Generalized minimum distance decoding," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 125–131, 1966.

[46] A. Sheikh, A. Graell i Amat, and G. Liva, "Binary message passing decoding of product codes based on generalized minimum distance decoding : (invited paper)," 03 2019, pp. 1–5.

[47] A. Sheikh, A. G. i Amat, G. Liva, C. Häger, and H. D. Pfister, "On low-complexity decoding of product codes for high-throughput fiber-optic systems," *CoRR*, vol. abs/1806.10903, 2018. [Online]. Available: http://arxiv.org/abs/1806.10903

[48] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, 1972.

[49] A. Graell i Amat, "Lecture notes for advanced topics in communications engineering," *TU Muenchen*, 2020.

[50] C. Jego and W. Gross, "Turbo decoding of product codes based on the modified adaptive belief propagation algorithm," 07 2007, pp. 641 – 644.

[51] C. Argon and S. W. McLaughlin, "An efficient chase decoder for turbo product codes," *IEEE Transactions on Communications*, vol. 52, no. 6, pp. 896–898, 2004.

[52] H. Mukhtar, A. Al-Dweik, and A. Shami, "Turbo product codes: Applications, challenges and future directions," *IEEE Communications Surveys and Tutorials*, vol. 18, pp. 1–1, 07 2016.

[53] G. Chen, L. Cao, L. Yu, and C.-W. Chen, "An efficient stopping criterion for turbo product codes," *Communications Letters, IEEE*, vol. 11, pp. 525 – 527, 07 2007.

[54] G. Lechner, T. Pedersen, and G. Kramer, "Analysis and design of binary message passing decoders," *IEEE Transactions on Communications*, vol. 60, no. 3, p. 601–607, Mar 2012. [Online]. Available: http://dx.doi.org/10.1109/TCOMM.2011.122111.100212

[55] G. Lechner and J. Sayir, "Improved sum-min decoding of ldpc codes," *International Symposium on Information Theory and its Applications, ISITA2004, Parma, Italy*, October 2004.

[56] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[57] F. Brannstrom, L. K. Rasmussen, and A. J. Grant, "Convergence analysis and optimal scheduling for multiple concatenated codes," *IEEE Transactions on Information Theory*, vol. 51, no. 9, pp. 3354–3364, 2005.

[58] J. Hagenauer, "The exit chart - introduction to extrinsic information transfer in iterative processing," in *2004 12th European Signal Processing Conference*, 2004, pp. 1541–1548.

[59] F. Abderrazak, M. Belkasmi, and A. Azouaoui, "Exit chart for iterative decoding of product and concatenated block codes," 07 2013.

[60] M. Fossorier and S. Lin, "Error performance analysis for reliability-based decoding algorithms," *Information Theory, IEEE Transactions on*, vol. 48, pp. 287 – 293, 02 2002.

[61] F. Lehmann and G. M. Maggio, "Analysis of the iterative decoding of ldpc and product codes using the gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 49, pp. 2993–3000, 2003.

[62] M. C. Davey and D. J. C. MacKay, "Monte carlo simulations of infinite low density parity check codes over gf(q)," in *International Workshop on Optimal Codes and Related Topics (OC98) Bulgaria*, June 9-15 1998. [Online]. Available: https://www.inference.org.uk/is/papers/ldpc-oc98.html

[63] F. Steiner, "Coding for higher-order modulation and probabilistic shaping," Dissertation, Technische Universität München, München, 2020.