Karlsruhe Institute of Technology
Communications Engineering Lab
Prof. Dr.-Ing. Laurent Schmalen

Application of Optimization Algorithms for Channel Decoding

Bachelor's Thesis

Andreas Tsouchlos

Advisor	:	Prof. Dr.-Ing. Laurent Schmalen
Supervisor	:	Dr.-Ing. Holger Jäkel

Start date	:	24.10.2022
End date	:	24.04.2023

Declaration

With this statement I declare that I have independently completed the above bachelors's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Karlsruhe, 24.04.2023

Andreas Tsouchlos

Acknowledgements

I would like to thank Prof. Dr.-Ing. Laurent Schmalen for granting me the opportunity to write my bachelor's thesis at the Communications Engineering Lab, as well as all other members of the institute for their help and many productive discussions, and for creating a very pleasant environment to do research in.

I am very grateful to Dr.-Ing. Holger Jäkel for kindly providing me with his knowledge and many suggestions, and for his constructive criticism during the preparation of this work.

Special thanks also to Mai Anh Vu for her invaluable feedback and support during the entire undertaking that is this thesis.

Finally, I would like to thank my family, who have enabled me to pursue my studies in a field I thoroughly enjoy and who have supported me completely throughout my journey.

Contents

1 Introduction

Channel coding using binary linear codes is a way of enhancing the reliability of data by detecting and correcting any errors that may occur during its transmission or storage. One class of binary linear codes, *low-density parity-check* (LDPC) codes, has become especially popular due to being able to reach arbitrarily small probabilities of error at code rates up to the capacity of the channel [Mac99, Sec. II.B.], while retaining a structure that allows for very efficient decoding. While the established decoders for LDPC codes, such as *belief propagation* (BP) and the *min-sum algorithm*, offer good decoding performance, they are suboptimal in most cases and exhibit an *error floor* for high *signal-to-noise ratio*s (SNRs) [RL09, Sec. 15.3], making them unsuitable for applications with extreme reliability requirements.

Optimization based decoding algorithms are an entirely different way of approaching the decoding problem. The first introduction of optimization techniques as a way of decoding binary linear codes was conducted in Feldman's 2003 Ph.D. thesis and a subsequent paper, establishing the field of *linear programming* (LP) decoding [Fel03], [FWK05]. There, the *maximum likelihood* (ML) decoding problem is approximated by a *linear program*, i.e., a linear, convex optimization problem, which can subsequently be solved using several different algorithms [TS06], [Von08], [Bar+13], [Zha+19]. More recently, novel approaches such as *proximal decoding* have been introduced. Proximal decoding is based on a non-convex optimization formulation of the *maximum a posteriori* (MAP) decoding problem [WT22].

The motivation behind applying optimization methods to channel decoding is to utilize existing techniques in the broad field of optimization theory, as well as to find new decoding methods not suffering from the same disadvantages as existing message passing based approaches or exhibiting other desirable properties. LP decoding, for example, comes with strong theoretical guarantees allowing it to be used as a way of closely approximating ML decoding [Bar+13, Sec. I], and proximal decoding is applicable to non-trivial channel models such as LDPC-coded massive *multiple-input multiple-output* (MIMO) channels [WT22].

This thesis aims to further the analysis of optimization based decoding algorithms as well as to verify and complement the considerations present in the existing literature. Specifically, the proximal decoding algorithm and LP decoding using the *alternating direction method of multipliers* (ADMM) [Bar+13] are explored within the context of *binary phase-shift keying* (BPSK) modulated *additive white Gaussian noise* (AWGN) channels. Implementations of both decoding methods are produced, and based on simulation results from those implementations the algorithms are examined and compared. Approaches to determine the optimal value of each parameter are derived and the computational and decoding performance of the algorithms is examined. An improvement on proximal decoding is suggested, achieving up to 1 dB of gain, depending on the parameters chosen and the code considered.

2 Theoretical Background

In this chapter, the theoretical background necessary to understand the decoding algorithms examined in this work is given. First, the notation used is clarified. The physical layer is detailed - the used modulation scheme and channel model. A short introduction to channel coding with binary linear codes and especially LDPC codes is given. The established methods of decoding LDPC codes are briefly explained. Lastly, the general process of decoding using optimization techniques is described and an overview of the utilized optimization methods is given.

2.1 General Remarks on Notation

Wherever the domain of a variable is expanded, this will be indicated with a tilde. For example:

$$\begin{aligned} x \in \{-1, 1\} &\to \tilde{x} \in \mathbb{R} \\ c \in \mathbb{F}_2 &\to \tilde{c} \in [0, 1] \subseteq \mathbb{R}. \end{aligned}$$

Additionally, a shorthand notation will be used, denoting a set of indices as

$$[m : n] := \{m, m+1, \ldots, n-1, n\}, \qquad m < n, \;\; m, n \in \mathbb{Z}.$$

In order to designate element-wise operations, in particular the *Hadamard product* and the *Hadamard power*, the operator $\circ$ will be used:

$$\begin{aligned} \boldsymbol{a} \circ \boldsymbol{b} &:= \begin{bmatrix} a_1 b_1 & \ldots & a_n b_n \end{bmatrix}^{\mathrm{T}}, \qquad && \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n, \;\; n \in \mathbb{N} \\ \boldsymbol{a}^{\circ k} &:= \begin{bmatrix} a_1^k \ldots a_n^k \end{bmatrix}^{\mathrm{T}}, && \boldsymbol{a} \in \mathbb{R}^n, \;\; n \in \mathbb{N}, k \in \mathbb{Z}. \end{aligned}$$

2.2 Channel Model and Modulation

In order to transmit a bit-word $\boldsymbol{c} \in \mathbb{F}_2^n$ of length n over a channel, it has to be mapped onto a symbol $\boldsymbol{x} \in \mathbb{R}^n$ that can be physically transmitted. This is known as modulation. The modulation scheme chosen here is BPSK:

$$\boldsymbol{x} = 1 - 2\boldsymbol{c}.$$

The transmitted symbol is distorted by the channel and denoted as $\boldsymbol{y} \in \mathbb{R}^n$. This distortion is described by the channel model, which in the context of this thesis is chosen to be AWGN:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}, \qquad n_i \in \mathcal{N}\left(0, \frac{\sigma^2}{2}\right), \;\; i \in [1 : n].$$

2.3 Channel Coding with LDPC Codes

Channel coding describes the process of adding redundancy to information transmitted over a channel in order to detect and correct any errors that may occur during the transmission. Encoding the information using *binary linear codes* is one way of conducting this process, whereby *data words* are mapped onto longer *codewords*, which carry redundant information. LDPC codes have become especially popular, since they are able to reach arbitrarily small probabilities of error at code rates up to the capacity of the channel [Mac99, Sec. II.B.], while having a structure that allows for very efficient decoding.

The lengths of the data words and codewords are denoted by $k \in \mathbb{N}$ and $n \in \mathbb{N}$, respectively, with $k \le n$. The set of codewords $\mathcal{C} \subset \mathbb{F}_2^n$ of a binary linear code can be represented using the *parity-check matrix* $\boldsymbol{H} \in \mathbb{F}_2^{m \times n}$, where m represents the number of parity-checks:

$$\mathcal{C} := \left\{ \boldsymbol{c} \in \mathbb{F}_2^n : \boldsymbol{H}\boldsymbol{c}^\text{T} = \boldsymbol{0} \right\}.$$

A data word $\boldsymbol{u} \in \mathbb{F}_2^k$ can be mapped onto a codeword $\boldsymbol{c} \in \mathbb{F}_2^n$ using the *generator matrix* $\boldsymbol{G} \in \mathbb{F}_2^{k \times n}$:

$$\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G}.$$

After obtaining a codeword from a data word, it is transmitted over a channel as described in section 2.2. The received signal $\boldsymbol{y}$ is then decoded to obtain an estimate of the transmitted codeword, denoted as $\hat{\boldsymbol{c}}$. Finally, the encoding procedure is reversed and an estimate of the originally sent data word, $\hat{\boldsymbol{u}}$, is produced. The methods examined in this work are all based on *soft-decision* decoding, i.e., $\boldsymbol{y}$ is considered to be in $\mathbb{R}^n$ and no preliminary decision is made by a demodulator. The process of transmitting and decoding a codeword is visualized in figure 2.1.

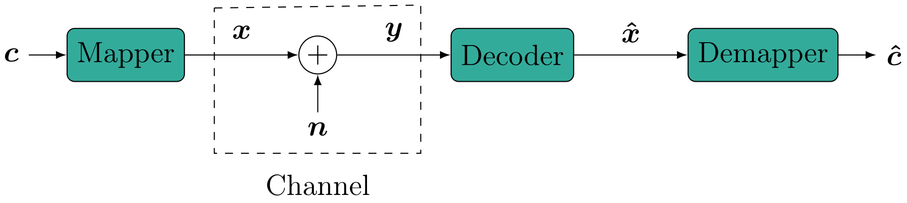


Figure 2.1: Overview of channel model and modulation

The decoding process itself is generally based either on the MAP or the ML criterion:

$$\hat{\boldsymbol{c}}_\text{MAP} = \underset{\boldsymbol{c} \in \mathcal{C}}{\operatorname{argmax}}\, p_{\boldsymbol{C}|\boldsymbol{Y}}\left(\boldsymbol{c} \mid \boldsymbol{y}\right)$$

$$\hat{\boldsymbol{c}}_\text{ML} = \underset{\boldsymbol{c} \in \mathcal{C}}{\operatorname{argmax}}\, f_{\boldsymbol{Y}|\boldsymbol{C}}\left(\boldsymbol{y} \mid \boldsymbol{c}\right).$$

The two criteria are closely connected through Bayes' theorem and are equivalent when the prior probability of transmitting a codeword is the same for all codewords:

$$\begin{aligned}\underset{c \in \mathcal{C}}{\operatorname{argmax}}\, p_{\boldsymbol{C}|\boldsymbol{Y}}\left(\boldsymbol{c} \mid \boldsymbol{y}\right) &= \underset{c \in \mathcal{C}}{\operatorname{argmax}} \frac{f_{\boldsymbol{Y}|\boldsymbol{C}}\left(\boldsymbol{y} \mid \boldsymbol{c}\right) p_{\boldsymbol{C}}\left(\boldsymbol{c}\right)}{f_{\boldsymbol{Y}}\left(\boldsymbol{y}\right)} \\ &= \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, f_{\boldsymbol{Y}|\boldsymbol{C}}\left(\boldsymbol{y} \mid \boldsymbol{c}\right).\end{aligned}$$

2.4 Tanner Graphs and Belief Propagation

It is often helpful to visualize codes graphically. This is especially true for LDPC codes, as the established decoding algorithms are *message passing algorithms*, which are inherently graph-based.

A binary linear code with a parity-check matrix $\boldsymbol{H}$ can be visualized using a *Tanner* or *factor graph*: Each row of $\boldsymbol{H}$, which represents one parity-check, is viewed as a *check node* (CN). Each component of the codeword $\boldsymbol{c}$ is interpreted as a *variable node* (VN). The relationship between CNs and VNs can then be plotted by noting which components of $\boldsymbol{c}$ are considered for which parity-check. Figure 2.2 shows the Tanner graph for the (7,4) Hamming code, which has the following parity-check matrix [RL09, Example 5.7.]:

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

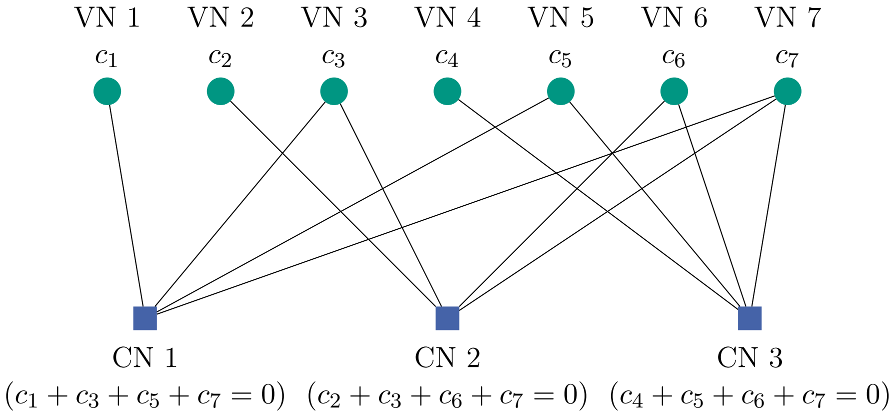

VN 1
VN 2
VN 3
VN 4
VN 5
VN 6
VN 7
c_1
c_2
c_3
c_4
c_5
c_6
c_7
CN 1
$(c_1 + c_3 + c_5 + c_7 = 0)$
CN 2
$(c_2 + c_3 + c_6 + c_7 = 0)$
CN 3
$(c_4 + c_5 + c_6 + c_7 = 0)$

Figure 2.2: Tanner graph for the (7,4) Hamming code

CNs and VNs, and by extension the rows and columns of $\boldsymbol{H}$, are indexed with the variables j and i. The sets of all CNs and all VNs are denoted by $\mathcal{J} := [1 : m]$ and $\mathcal{I} := [1 : n]$, respectively. The *neighborhood* of the jth CN, i.e., the set of all adjacent VNs, is denoted by $N_c\left(j\right)$. The neighborhood of the ith VN is denoted by $N_v\left(i\right)$. For the code depicted in figure 2.2, for example, $N_c\left(1\right) = \{1, 3, 5, 7\}$ and $N_v\left(3\right) = \{1, 2\}$. The degree d_j of a CN is defined as the number of adjacent VNs: $d_j := \left|N_c\left(j\right)\right|$; the degree of a VN is similarly defined as $d_i := \left|N_v\left(i\right)\right|$.

Message passing algorithms are based on the notion of passing messages between CNs and VNs. BP is one such algorithm that is commonly used to decode LDPC codes. It aims to compute the posterior probabilities $p_{C_i|\boldsymbol{Y}}\left(c_i = 1|\boldsymbol{y}\right), \;\; i \in \mathcal{I}$, see [Mac99, Sec. III.] and use them to calculate the estimate $\hat{\boldsymbol{c}}$. For cycle-free graphs this goal is reached after a finite number of steps and BP is equivalent to MAP decoding. When the graph contains cycles, however, BP only approximates the MAP probabilities and is sub-optimal. This leads to generally worse performance than MAP decoding for practical codes. Additionally, an *error floor* appears for very high SNRs, making the use of BP impractical for applications where a very low error rate is desired [RL09, Sec. 15.3]. Another popular decoding method for LDPC codes is the *min-sum algorithm*. This is a simplification of BP using an approximation of the non-linear tanh function to improve the computational performance.

2.5 Decoding using Optimization Methods

The general idea behind using optimization methods for channel decoding is to reformulate the decoding problem as an optimization problem. This new formulation can then be solved with one of the many available optimization algorithms.

Generally, the original decoding problem considered is either the MAP or the ML decoding problem:

$$\hat{\boldsymbol{c}}_{\text{MAP}} = \operatorname*{argmax}_{\boldsymbol{c}\in\mathcal{C}} p_{\boldsymbol{C}|\boldsymbol{Y}} \left(\boldsymbol{c} \mid \boldsymbol{y}\right) \tag{2.1}$$

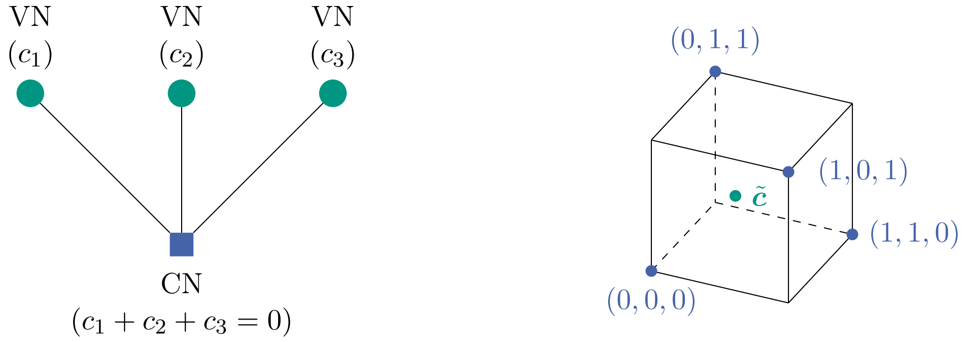
$$\hat{\boldsymbol{c}}_{\text{ML}} = \operatorname*{argmax}_{\boldsymbol{c}\in\mathcal{C}} f_{\boldsymbol{Y}|\boldsymbol{C}} \left(\boldsymbol{y} \mid \boldsymbol{c}\right). \tag{2.2}$$

The goal is to arrive at a formulation, where a certain objective function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ must be minimized under certain constraints:

$$\begin{aligned} \text{minimize}\quad & g\left(\tilde{\boldsymbol{c}}\right) \\ \text{subject to}\quad & \tilde{\boldsymbol{c}} \in D, \end{aligned}$$

where $D \subseteq \mathbb{R}^n$ is the domain of values attainable for $\tilde{\boldsymbol{c}}$ and represents the constraints under which the minimization is to take place.

In contrast to the established message-passing decoding algorithms, the perspective then changes from observing the decoding process in its Tanner graph representation with VNs and CNs (as shown in figure 2.3a) to a spatial representation (figure 2.3b), where the codewords are some of the vertices of a hypercube. The goal is to find the point $\tilde{\boldsymbol{c}}$, which minimizes the objective function g.



(a) Tanner graph representation of a single parity-check code

(b) Spatial representation of a single parity-check code

Figure 2.3: Different representations of the decoding problem

2.6 A Short Introduction to the Proximal Gradient Method and ADMM

In this section, the general ideas behind the optimization methods used in this work are outlined. The application of these optimization methods to channel decoding decoding will be discussed in later chapters. Two methods are introduced, the *proximal gradient method* and ADMM.

Proximal algorithms are algorithms for solving convex optimization problems that rely on the use of *proximal operators*. The proximal operator $\mathbf{prox}_{\lambda f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is defined by [PB14, Sec. 1.1]

$$\mathbf{prox}_{\lambda f}\left(\boldsymbol{v}\right) = \underset{\boldsymbol{x} \in \mathbb{R}^n}{\operatorname{argmin}} \left(f\left(\boldsymbol{x}\right) + \frac{1}{2\lambda} \lVert \boldsymbol{x} - \boldsymbol{v} \rVert_2^2 \right).$$

This operator computes a point that is a compromise between minimizing f and staying in the proximity of $\boldsymbol{v}$. The parameter λ determines how each term is weighed. The proximal gradient method is an iterative optimization method utilizing proximal operators, used to solve problems of the form

$$\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad f\left(\boldsymbol{x}\right) + g\left(\boldsymbol{x}\right)$$

that consists of two steps: minimizing f with gradient descent and minimizing g using the proximal operator [PB14, Sec. 4.2]:

$$\begin{aligned} \boldsymbol{x} &\leftarrow \boldsymbol{x} - \lambda \nabla f\left(\boldsymbol{x}\right) \\ \boldsymbol{x} &\leftarrow \mathbf{prox}_{\lambda g}\left(\boldsymbol{x}\right), \end{aligned}$$

Since g is minimized with the proximal operator and is thus not required to be differentiable, it can be used to encode the constraints of the optimization problem (e.g., in the form of an *indicator function*, as mentioned in [PB14, Sec. 1.2]).

ADMM is another optimization method. In this thesis it will be used to solve a *linear program*, which is a special type of convex optimization problem in which the objective function is linear and the constraints consist of linear equalities and inequalities. Generally, any linear program can be expressed in *standard form*[1] [BT97, Sec. 1.1]:

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{\gamma}^{\mathrm{T}} \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \mathbf{0}, \end{aligned} \tag{2.3}$$

where $\boldsymbol{x}, \boldsymbol{\gamma} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. A technique called *Lagrangian relaxation* can then be applied [BT97, Sec. 11.4]. First, some of the constraints are moved into the objective function itself and weights $\boldsymbol{\lambda}$ are introduced. A new, relaxed problem is formulated as

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{\gamma}^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}} \left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right) \\ \text{subject to} \quad & \boldsymbol{x} \geq \mathbf{0}, \end{aligned} \tag{2.4}$$

[1] The inequality $\boldsymbol{x} \geq \mathbf{0}$ is to be interpreted componentwise.

the new objective function being the *Lagrangian*[2]

$$\mathcal{L}\left(\boldsymbol{x}, \boldsymbol{\lambda}\right) = \boldsymbol{\gamma}^{\text{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\text{T}}\left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right).$$

This problem is not directly equivalent to the original one, as the solution now depends on the choice of the *Lagrange multipliers* $\boldsymbol{\lambda}$. Interestingly, however, for this particular class of problems, the minimum of the objective function (hereafter called *optimal objective*) of the relaxed problem (2.4) is a lower bound for the optimal objective of the original problem (2.3) [BT97, Sec. 4.1]:

$$\min_{\boldsymbol{x} \ge \boldsymbol{0}} \mathcal{L}\left(\boldsymbol{x}, \boldsymbol{\lambda}\right) \le \min_{\substack{\boldsymbol{x} \ge \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}}} \boldsymbol{\gamma}^{\text{T}}\boldsymbol{x}.$$

Furthermore, for uniquely solvable linear programs *strong duality* always holds [BT97, Theorem 4.4]. This means that not only is it a lower bound, the tightest lower bound actually reaches the value itself: in other words, with the optimal choice of $\boldsymbol{\lambda}$, the optimal objectives of the problems (2.4) and (2.3) have the same value, i.e.,

$$\max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \min_{\boldsymbol{x} \ge \boldsymbol{0}} \mathcal{L}\left(\boldsymbol{x}, \boldsymbol{\lambda}\right) = \min_{\substack{\boldsymbol{x} \ge \boldsymbol{0} \\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}}} \boldsymbol{\gamma}^{\text{T}}\boldsymbol{x}.$$

Thus, we can define the *dual problem* as the search for the tightest lower bound:

$$\underset{\boldsymbol{\lambda} \in \mathbb{R}^m}{\text{maximize}} \quad \min_{\boldsymbol{x} \ge \boldsymbol{0}} \mathcal{L}\left(\boldsymbol{x}, \boldsymbol{\lambda}\right), \tag{2.5}$$

and recover the solution $\boldsymbol{x}_{\text{opt}}$ to problem (2.3) from the solution $\boldsymbol{\lambda}_{\text{opt}}$ to problem (2.5) by computing [Boy+11, Sec. 2.1]

$$\boldsymbol{x}_{\text{opt}} = \underset{\boldsymbol{x} \ge \boldsymbol{0}}{\text{argmin}}\, \mathcal{L}\left(\boldsymbol{x}, \boldsymbol{\lambda}_{\text{opt}}\right). \tag{2.6}$$

The dual problem can then be solved iteratively using *dual ascent*: starting with an initial estimate for $\boldsymbol{\lambda}$, calculate an estimate for $\boldsymbol{x}$ using equation (2.6); then, update $\boldsymbol{\lambda}$ using gradient descent [Boy+11, Sec. 2.1]:

$$\begin{aligned} \boldsymbol{x} &\leftarrow \underset{\boldsymbol{x} \ge \boldsymbol{0}}{\text{argmin}}\, \mathcal{L}\left(\boldsymbol{x}, \boldsymbol{\lambda}\right) \\ \boldsymbol{\lambda} &\leftarrow \boldsymbol{\lambda} + \alpha\left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right), \qquad \alpha > 0. \end{aligned}$$

The algorithm can be improved by observing that when the objective function $g : \mathbb{R}^n \to \mathbb{R}$ is separable into a sum of $N \in \mathbb{N}$ sub-functions $g_i : \mathbb{R}^{n_i} \to \mathbb{R}$, i.e., $g\left(\boldsymbol{x}\right) = \sum_{i=1}^{N} g_i\left(\boldsymbol{x}_i\right)$, where $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$, $i \in [1 : N]$ are subvectors of $\boldsymbol{x}$, the Lagrangian is as well:

$$\begin{aligned} &\text{minimize} \quad && \sum_{i=1}^{N} g_i\left(\boldsymbol{x}_i\right) \\ &\text{subject to} \quad && \sum_{i=1}^{N} \boldsymbol{A}_i\boldsymbol{x}_i = \boldsymbol{b} \end{aligned}$$

[2] Depending on what literature is consulted, the definition of the Lagrangian differs in the order of $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{b}$. As will subsequently be seen, however, the only property of the Lagrangian having any bearing on the optimization process is that minimizing it gives a lower bound on the optimal objective of the original problem. This property is satisfied no matter the order of the terms and the order chosen here is the one used in the LP decoding literature making use of ADMM.

$$\mathcal{L}\left(\left(\boldsymbol{x}_i\right)_{i=1}^N, \boldsymbol{\lambda}\right) = \sum_{i=1}^{N} g_i\left(\boldsymbol{x}_i\right) + \boldsymbol{\lambda}^\text{T}\left(\sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{x}_{\boldsymbol{i}} - \boldsymbol{b}\right).$$

The matrices $\boldsymbol{A}_i \in \mathbb{R}^{m \times n_i}$, $i \in [1:N]$ form a partition of $\boldsymbol{A}$, corresponding to $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & \ldots & \boldsymbol{A}_N \end{bmatrix}$. The minimization of each term can happen in parallel, in a distributed fashion [Boy+11, Sec. 2.2]. In each minimization step, only one subvector $\boldsymbol{x}_i$ of $\boldsymbol{x}$ is considered, regarding all other subvectors as being constant. This modified version of dual ascent is called *dual decomposition*:

$$\begin{aligned} \boldsymbol{x}_i &\leftarrow \operatorname*{argmin}_{\boldsymbol{x}_i \ge \boldsymbol{0}} \mathcal{L}\left(\left(\boldsymbol{x}_i\right)_{i=1}^N, \boldsymbol{\lambda}\right) \qquad \forall i \in [1:N] \\ \boldsymbol{\lambda} &\leftarrow \boldsymbol{\lambda} + \alpha\left(\sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{b}\right), \qquad \alpha > 0. \end{aligned}$$

ADMM works the same way as dual decomposition. It only differs in the use of an *augmented Lagrangian* $\mathcal{L}_\mu\left(\left(\boldsymbol{x}\right)_{i=1}^N, \boldsymbol{\lambda}\right)$ in order to strengthen the convergence properties. The augmented Lagrangian extends the classical one with an additional penalty term with the penalty parameter μ:

$$\mathcal{L}_\mu\left(\left(\boldsymbol{x}\right)_{i=1}^N, \boldsymbol{\lambda}\right) = \underbrace{\sum_{i=1}^{N} g_i\left(\boldsymbol{x}_{\boldsymbol{i}}\right) + \boldsymbol{\lambda}^\text{T}\left(\sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{b}\right)}_{\text{Classical Lagrangian}} + \underbrace{\frac{\mu}{2}\left\lVert \sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{b} \right\rVert_2^2}_{\text{Penalty term}}, \qquad \mu > 0.$$

The steps to solve the problem are the same as with dual decomposition, with the added condition that the step size be μ:

$$\begin{aligned} \boldsymbol{x}_i &\leftarrow \operatorname*{argmin}_{\boldsymbol{x}_i \ge \boldsymbol{0}} \mathcal{L}_\mu\left(\left(\boldsymbol{x}\right)_{i=1}^N, \boldsymbol{\lambda}\right) \qquad \forall i \in [1:N] \\ \boldsymbol{\lambda} &\leftarrow \boldsymbol{\lambda} + \mu\left(\sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{b}\right), \qquad \mu > 0. \end{aligned}$$

In subsequent chapters, the decoding problem will be reformulated as an optimization problem using two different methodologies. In chapter 3, a non-convex optimization approach is chosen and addressed using the proximal gradient method. In chapter 4, an LP based optimization problem is formulated and solved using ADMM.

3 Proximal Decoding

In this chapter, the proximal decoding algorithm is examined. First, the algorithm itself is described and some useful considerations concerning the implementation are presented. Simulation results are shown, based on which behavior of the algorithm is investigated for different codes and parameters. Finally, an improvement on proximal decoding is proposed.

3.1 Decoding Algorithm and Implementation

Proximal decoding was proposed by Wadayama et al. as a novel formulation of optimization-based decoding [WT22]. With this algorithm, minimization is performed using the proximal gradient method. In contrast to LP decoding, which will be covered in chapter 4, the objective function is based on a non-convex optimization formulation of the MAP decoding problem.

In order to derive the objective function, the authors begin with the MAP decoding rule, expressed as a continuous maximization problem[1] over $\boldsymbol{x}$:

$$\hat{\boldsymbol{x}} = \underset{\tilde{\boldsymbol{x}} \in \mathbb{R}^n}{\operatorname{argmax}} f_{\tilde{\boldsymbol{X}} \mid \boldsymbol{Y}} \left(\tilde{\boldsymbol{x}} \mid \boldsymbol{y} \right) = \underset{\tilde{\boldsymbol{x}} \in \mathbb{R}^n}{\operatorname{argmax}} f_{\boldsymbol{Y} \mid \tilde{\boldsymbol{X}}} \left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}} \right) f_{\tilde{\boldsymbol{X}}} \left(\tilde{\boldsymbol{x}} \right). \tag{3.1}$$

The likelihood $f_{\boldsymbol{Y} \mid \tilde{\boldsymbol{X}}} \left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}} \right)$ is a known function determined by the channel model. The prior PDF $f_{\tilde{\boldsymbol{X}}} \left(\tilde{\boldsymbol{x}} \right)$ is also known, as the equal probability assumption is made on $\mathcal{C}$. However, since the considered domain is continuous, the prior PDF cannot be ignored as a constant during the minimization as is often done, and has a rather unwieldy representation:

$$f_{\tilde{\boldsymbol{X}}} \left(\tilde{\boldsymbol{x}} \right) = \frac{1}{|\mathcal{C}|} \sum_{\boldsymbol{c} \in \mathcal{C}} \delta\big(\tilde{\boldsymbol{x}} - (-1)^{\boldsymbol{c}} \big). \tag{3.2}$$

In order to rewrite the prior PDF $f_{\tilde{\boldsymbol{X}}} \left(\tilde{\boldsymbol{x}} \right)$, the so-called *code-constraint polynomial* is introduced as

$$h \left(\tilde{\boldsymbol{x}} \right) = \underbrace{\sum_{i=1}^{n} \left(\tilde{x}_i^{\,2} - 1 \right)^2}_{\text{Bipolar constraint}} + \underbrace{\sum_{j=1}^{m} \left[\left(\prod_{i \in N_c(j)} \tilde{x}_i \right) - 1 \right]^2}_{\text{Parity constraint}}.$$

The intention of this function is to provide a way to penalize vectors far from a codeword and favor those close to one. In order to achieve this, the polynomial is composed of two parts: one term representing the bipolar constraint, providing for a discrete solution of the continuous optimization problem, and one term representing the parity constraints, accommodating the role of the parity-check matrix $\boldsymbol{H}$. The prior PDF is then approximated

[1] The expansion of the domain to be continuous doesn't constitute a material difference in the meaning of the rule. The only change is that what previously were *probability mass functions* (PMFs) now have to be expressed in terms of *probability density functions* (PDFs).

using the code-constraint polynomial as:

$$f_{\tilde{\boldsymbol{X}}}\left(\tilde{\boldsymbol{x}}\right) \approx \frac{1}{Z}\mathrm{e}^{-\gamma h(\tilde{\boldsymbol{x}})}. \tag{3.3}$$

The authors justify this approximation by arguing that for $\gamma \to \infty$, the approximation in equation (3.3) approaches the original function in equation (3.2). This approximation can then be plugged into equation (3.1) and the likelihood can be rewritten using the negative log-likelihood $L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right) = -\ln\left(f_{\boldsymbol{Y}|\tilde{\boldsymbol{X}}}\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right)\right)$ as

$$\begin{aligned}\hat{\boldsymbol{x}} &= \underset{\tilde{\boldsymbol{x}}\in\mathbb{R}^n}{\operatorname{argmax}}\, \mathrm{e}^{-L(\boldsymbol{y}|\tilde{\boldsymbol{x}})}\mathrm{e}^{-\gamma h(\tilde{\boldsymbol{x}})} \\ &= \underset{\tilde{\boldsymbol{x}}\in\mathbb{R}^n}{\operatorname{argmin}}\, L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right) + \gamma h\left(\tilde{\boldsymbol{x}}\right).\end{aligned}$$

Thus, with proximal decoding, the objective function $g\left(\tilde{\boldsymbol{x}}\right)$ considered is

$$g\left(\tilde{\boldsymbol{x}}\right) = L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right) + \gamma h\left(\tilde{\boldsymbol{x}}\right) \tag{3.4}$$

and the decoding problem is reformulated to

$$\begin{aligned}\text{minimize}\ \ & L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right) + \gamma h\left(\tilde{\boldsymbol{x}}\right) \\ \text{subject to}\ \ & \tilde{\boldsymbol{x}} \in \mathbb{R}^n.\end{aligned}$$

For the solution of the approximate MAP decoding problem using the proximal gradient method, the two parts of equation (3.4) are considered separately: the minimization of the objective function occurs in an alternating fashion, switching between the negative log-likelihood $L\left(\boldsymbol{y} \mid \boldsymbol{x}\right)$ and the scaled code-constraint polynomial $\gamma h\left(\boldsymbol{x}\right)$. Two helper variables, $\boldsymbol{r}$ and $\boldsymbol{s}$, are introduced, describing the result of each of the two steps. The first step, minimizing the log-likelihood, is performed using gradient descent:

$$\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega\nabla L\left(\boldsymbol{y} \mid \boldsymbol{s}\right), \qquad \omega > 0. \tag{3.5}$$

For the second step, minimizing the scaled code-constraint polynomial, the *proximal operator* of $\gamma h\left(\tilde{\boldsymbol{x}}\right)$ has to be computed. It is then immediately approximated with gradient-descent:

$$\begin{aligned}\mathbf{prox}_{\gamma h}\left(\tilde{\boldsymbol{x}}\right) &\equiv \underset{\boldsymbol{t}\in\mathbb{R}^n}{\operatorname{argmin}}\, \gamma h\left(\boldsymbol{t}\right) + \frac{1}{2}\left\lVert \boldsymbol{t} - \tilde{\boldsymbol{x}}\right\rVert \\ &\approx \tilde{\boldsymbol{x}} - \gamma\nabla h\left(\tilde{\boldsymbol{x}}\right), \qquad \gamma > 0,\ \text{small}.\end{aligned}$$

The second optimization step thus becomes

$$\boldsymbol{s} \leftarrow \boldsymbol{r} - \gamma\nabla h\left(\boldsymbol{r}\right), \qquad \gamma > 0,\ \text{small}.$$

While the approximation of the prior PDF made in equation (3.3) theoretically becomes better with larger γ, the constraint that γ be small is important, as it keeps the effect of $h\left(\tilde{\boldsymbol{x}}\right)$ on the landscape of the objective function small. Otherwise, unwanted stationary points, including local minima, are introduced. The authors say that "[. . .] in practice, the value of γ should be adjusted according to the decoding performance" [WT22, Sec. 3.1].

In the case of AWGN, the likelihood $f_{\boldsymbol{Y}|\tilde{\boldsymbol{X}}}\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right)$ is

$$f_{\boldsymbol{Y}|\tilde{\boldsymbol{X}}}\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\mathrm{e}^{-\frac{\lVert \boldsymbol{y}-\tilde{\boldsymbol{x}}\rVert^2}{2\sigma^2}}.$$

Thus, the gradient of the negative log-likelihood becomes[2]

$$\begin{aligned}\nabla L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right) &\propto -\nabla \lVert \boldsymbol{y} - \tilde{\boldsymbol{x}}\rVert^2 \\ &\propto \tilde{\boldsymbol{x}} - \boldsymbol{y},\end{aligned}$$

allowing equation (3.5) to be rewritten as

$$\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega\left(\boldsymbol{s} - \boldsymbol{y}\right).$$

One thing to consider during the actual decoding process, is that the gradient of the code-constraint polynomial can take on extremely large values. To avoid numerical instability, an additional step is added, where all components of the current estimate are clipped to $[-\eta, \eta]$, where η is a positive constant slightly larger than one:

$$\boldsymbol{s} \leftarrow \Pi_\eta\left(\boldsymbol{r} - \gamma\nabla h\left(\boldsymbol{r}\right)\right),$$

$\Pi_\eta\left(\cdot\right)$ expressing the projection onto $[-\eta, \eta]^n$.

The iterative decoding process resulting from these considerations is depicted in algorithm 3.1.

```
s ← 0
for $K$ iterations do
    $\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega\left(\boldsymbol{s} - \boldsymbol{y}\right)$
    $\boldsymbol{s} \leftarrow \Pi_\eta\left(\boldsymbol{r} - \gamma\nabla h\left(\boldsymbol{r}\right)\right)$
    $\hat{\boldsymbol{x}} \leftarrow \operatorname{sign}\left(\boldsymbol{s}\right)$
    if $\boldsymbol{H}\hat{\boldsymbol{c}} = \boldsymbol{0}$ do
        return $\hat{\boldsymbol{c}}$
    end if
end for
return $\hat{\boldsymbol{c}}$
```

Algorithm 3.1: Proximal decoding algorithm for an AWGN channel. Based on Algorithm 1 in [WT22]

The algorithm was first implemented in Python because of the fast development process and straightforward debugging ability. It was subsequently reimplemented in C++ using the Eigen[3] linear algebra library to achieve higher performance. The focus has been on a fast implementation, sometimes at the expense of memory usage, somewhat limiting the size of the codes the implementation can be used with. The evaluation of decoding

[2] For the minimization, constants can be disregarded. For this reason, it suffices to consider only proportionality instead of equality.

[3] `https://eigen.tuxfamily.org`

operations and subsequent calculation of *bit error rate*s (BERs), *frame error rate*s (FERs), etc., has been wholly realized in Python.

Concerning the proximal decoding algorithm itself, there are certain aspects presenting optimization opportunities during the implementation. The gradient of the code-constraint polynomial [WT22, Sec. 2.3], for example, is given by

$$\nabla h\left(\tilde{\boldsymbol{x}}\right) = \begin{bmatrix} \frac{\partial}{\partial \tilde{x}_1} h\left(\tilde{\boldsymbol{x}}\right) & \ldots & \frac{\partial}{\partial \tilde{x}_n} h\left(\tilde{\boldsymbol{x}}\right) & \end{bmatrix}^{\mathrm{T}},$$

$$\frac{\partial}{\partial \tilde{x}_k} h\left(\tilde{\boldsymbol{x}}\right) = 4\left(\tilde{x}_k^2 - 1\right)\tilde{x}_k + \frac{2}{\tilde{x}_k} \sum_{j \in N_v(k)} \left(\left(\prod_{i \in N_c(j)} \tilde{x}_i \right)^2 - \prod_{i \in N_c(j)} \tilde{x}_i \right).$$

Since the products $\prod_{i \in N_c(j)} \tilde{x}_i, \ j \in \mathcal{J}$ are identical for all components $\tilde{x}_k$ of $\tilde{\boldsymbol{x}}$, they can be precomputed. Defining

$$\boldsymbol{p} := \begin{bmatrix} \prod_{i \in N_c(1)} \tilde{x}_i \\ \vdots \\ \prod_{i \in N_c(m)} \tilde{x}_i \end{bmatrix} \quad \text{and} \quad \boldsymbol{v} := \boldsymbol{p}^{\circ 2} - \boldsymbol{p},$$

the gradient can be written as

$$\nabla h\left(\tilde{\boldsymbol{x}}\right) = 4\left(\tilde{\boldsymbol{x}}^{\circ 3} - \tilde{\boldsymbol{x}}\right) + 2\tilde{\boldsymbol{x}}^{\circ(-1)} \circ \boldsymbol{H}^{\mathrm{T}}\boldsymbol{v},$$

enabling its computation primarily with element-wise operations and matrix-vector multiplication. This is beneficial, as the libraries employed for the implementation are heavily optimized for such calculations (e.g., through vectorization of the operations).

The projection $\prod_\eta (.)$ also proves straightforward to compute, as it amounts to simply clipping each component of the vector onto $[-\eta, \eta]$ individually.

3.2 Analysis and Simulation Results

In this section, the general behavior of the proximal decoding algorithm is analyzed. The impact of the parameters γ, as well as ω, the maximum number of iterations K and η is examined. The decoding performance is assessed based on the BER and the FER as well as the *decoding failure rate* - the rate at which the algorithm produces results that are not valid codewords. The convergence properties are reviewed and related to the decoding performance. Finally, the computational performance is examined on a theoretical basis as well as on the basis of the implementation completed in the context of this thesis.

All simulation results presented hereafter are based on Monte Carlo simulations. The BER and FER curves in particular have been generated by producing at least 100 frame errors for each data point, unless otherwise stated.

3.2.1 Choice of Parameters

First, the effect of the parameter γ is investigated. Figure 3.1 shows a comparison of the decoding performance of the proximal decoding algorithm as presented by Wadayama et al. [WT22] and the implementation realized for this work. The BER curves for three different choices of γ are shown, as well as the curve resulting from decoding using a BP decoder, as a reference. The results from Wadayama et al. are shown with solid lines, while the newly generated ones are shown with dashed lines. The parameters chosen are $K = 200, \mu = 0.05$ and $\eta = 1.5$. It is noticeable that for a moderately chosen value of γ

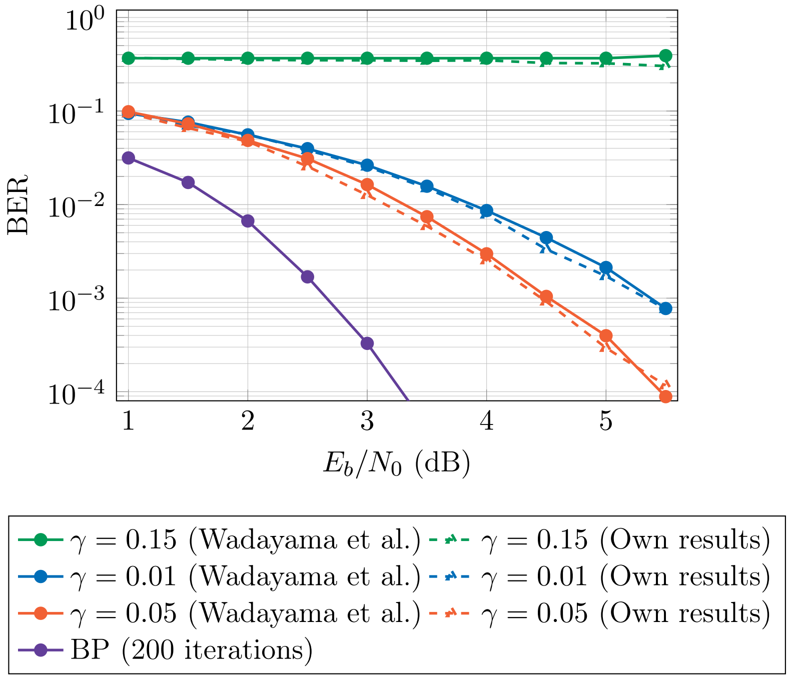


Figure 3.1: Comparison of datapoints from Wadayama et al. with own simulation results. (3, 6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

($\gamma = 0.05$) the decoding performance is better than for low ($\gamma = 0.01$) or high ($\gamma = 0.15$) values. The question arises whether there is some optimal value maximizing the decoding performance, especially since it seems to dramatically depend on γ. To better understand how they are related, figure 3.1 was recreated, but with a considerably larger selection of values for γ. In this new graph, shown in figure 3.2, instead of stacking the BER curves on top of one another in the same plot, the visualization is extended to three dimensions. The previously shown results are highlighted.

Evidently, while the decoding performance does depend on the value of γ, there is no single optimal value offering optimal performance, but rather a certain interval in which it stays largely unchanged. This indicates that while the choice of the parameter γ significantly affects the decoding performance, there is not much benefit attainable in undertaking an extensive search for an exact optimum. Rather, a preliminary examination providing a rough window for γ may be sufficient. Similar results can be observed when comparing several different codes, as shown in figure 3.13. It is apparent that while the exact landscape of the graph depends on the code, the general behavior is the same for all codes analyzed in this thesis.

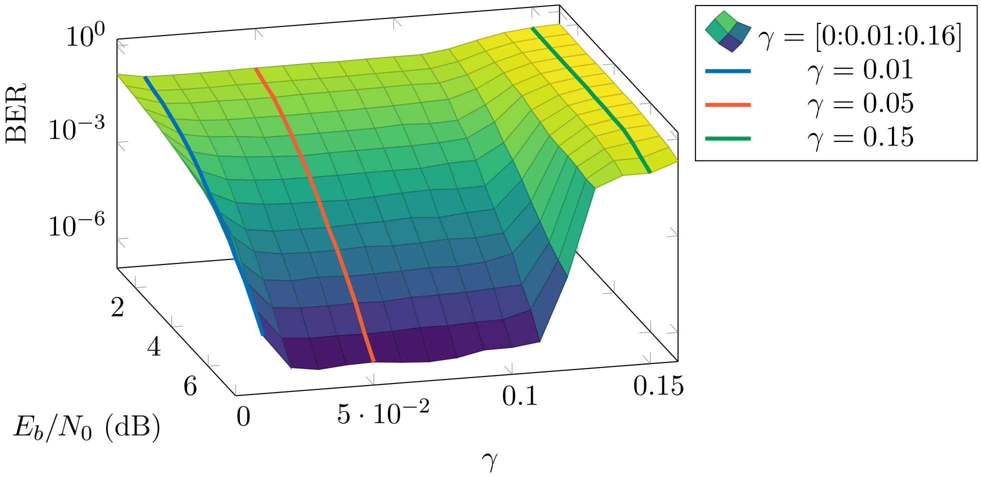


Figure 3.2: Visualization of the relationship between the decoding performance and the parameter γ. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

The parameter γ describes the step-size for the optimization step dealing with the code-constraint polynomial; the parameter ω describes the step-size for the step dealing with the negative-log likelihood. The relationship between ω and γ is portrayed in figure 3.3. The color of each cell indicates the BER when the corresponding values are chosen for the decoding. The SNR is kept constant at 4 dB. The BER exhibits similar behavior in its dependency on ω and on γ: it is minimized when keeping the value within certain bounds, without displaying a single distinct optimum. It is noteworthy that the decoder seems to achieve the best performance for similar values of the two step sizes. Again, this consideration applies to a multitude of different codes, as depicted in figure 3.14.

To better understand how to determine the optimal value for the parameter K, the average error is inspected. This time γ and ω are held constant at 0.05 and the average error is observed during each iteration of the decoding process, for several different SNRs. The plots have been generated by averaging the error over 500 000 decodings.

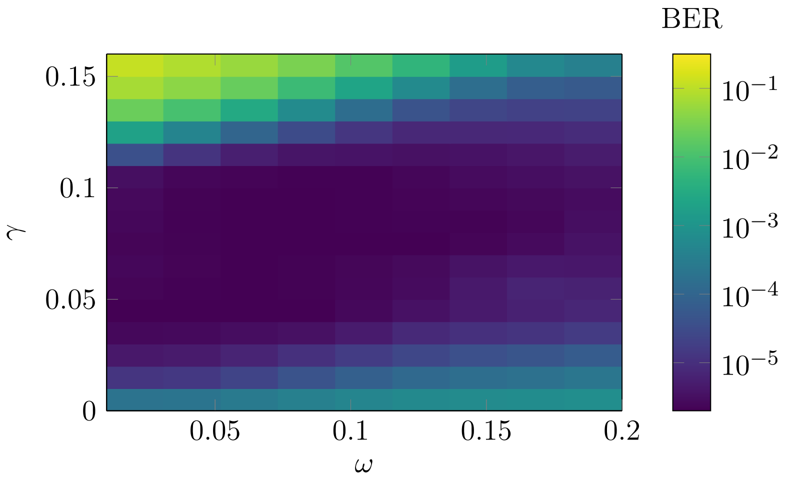


Figure 3.3: The BER as a function of the two step sizes. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

As some decodings go on for more iterations than others, the number of values which are averaged for each datapoints vary. This explains the dip visible in all curves around the 20th iteration, since after this point more and more correct decodings are completed, leaving more and more faulty ones to be averaged. Additionally, at this point the decline in the average error stagnates, rendering an increase in K counterproductive as it only raises the average timing requirements of the decoding process. Another aspect to consider is that the higher the SNR, the fewer decodings are present at each iteration to average, since a solution is found earlier. This explains the decreasing smoothness of the lines as the SNR increases. Remarkably, the SNR seems to not have any impact on the number of iterations necessary to reach the point at which the average error stabilizes.

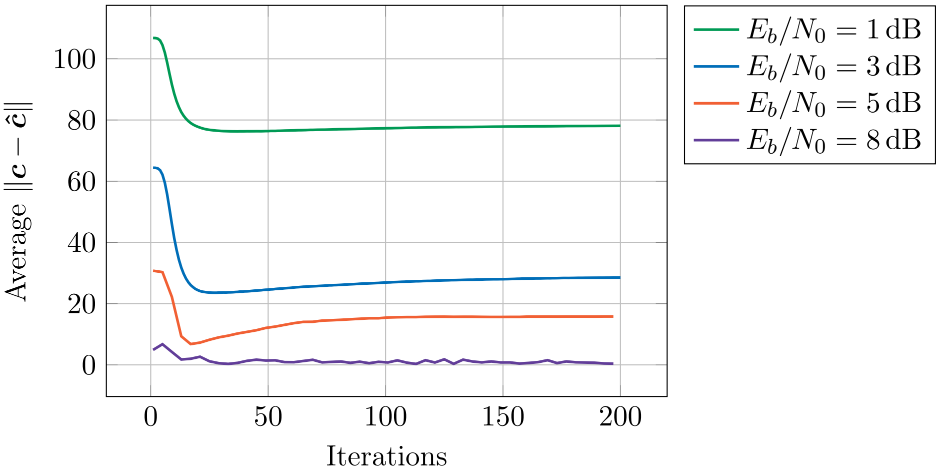


Figure 3.4: Average error for 500 000 decodings. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

Changing the parameter η does not appear to have a significant effect on the decoding performance when keeping the value within a reasonable window ("slightly larger than one", as stated in [WT22, Sec. 3.2]), which seems plausible considering its only function is ensuring numerical stability.

Summarizing the above considerations, an intricate strategy to find the exact optimum values for the parameters γ and ω appears to bring limited benefit; an initial rudimentary examination to find the general bounds in which the two values should lie is sufficient. The parameter K is independent of the SNR and raising its value above a certain threshold does not improve the decoding performance. The choice of η is insignificant and the parameter is only relevant as a means to bring about numerical stability.

3.2.2 Decoding Performance

Until now, only the BER has been considered to gauge the decoding performance. The FER, however, shows considerably different behavior, as can be seen in figure 3.5. Besides the BER and FER curves, the figure also shows the *decoding failure rate*. This is the rate at which the iterative process produces invalid codewords, i.e., the stopping criterion (line 6 of algorithm 3.1) is never satisfied and the maximum number of iterations K is reached without converging to a valid codeword. Three lines are plotted in each case, corresponding to different values of the parameter γ. The values chosen are the same as in figure 3.1, as they seem to adequately describe the behavior across a wide range of values (see figure 3.2).

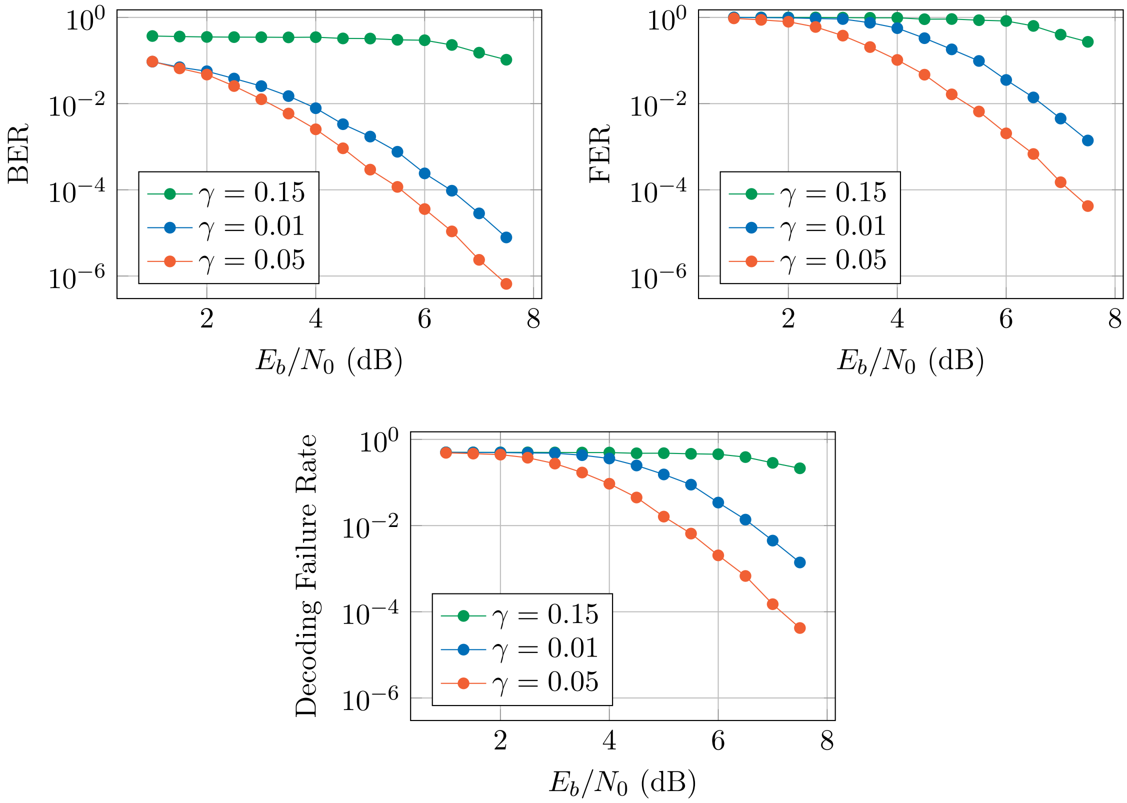


Figure 3.5: Comparison of FER, BER and decoding failure rate. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

It is apparent that the FER and the decoding failure rate are extremely similar, especially for higher SNRs. This leads to the hypothesis that, at least for higher SNRs, frame errors arise mainly due to the non-convergence of the algorithm instead of convergence to the wrong codeword. This course of thought will be picked up in section 3.3 when proposing a method to improve the algorithm.

In summary, the BER and FER indicate dissimilar decoding performance. The decoding failure rate closely resembles the FER, suggesting that the frame errors may largely be attributed to decoding failures.

3.2.3 Convergence Properties

The previous observation that the FER may arise mainly due to the non-convergence of the algorithm instead of convergence to the wrong codeword, raises the question why the decoding process does not converge so often. To better understand this issue, the iterative process is visualized in figure 3.6. In order to be able to simultaneously consider all components of the vectors being dealt with, a BCH code with $n = 7$ and $k = 4$ is chosen. Each plot shows one component of the current estimate during a given iteration ($\boldsymbol{r}$ and $\boldsymbol{s}$ are counted as different estimates and their values are interwoven to obtain the shown result), as well as the gradients of the negative log-likelihood and the code-constraint polynomial, which influence the next estimate.

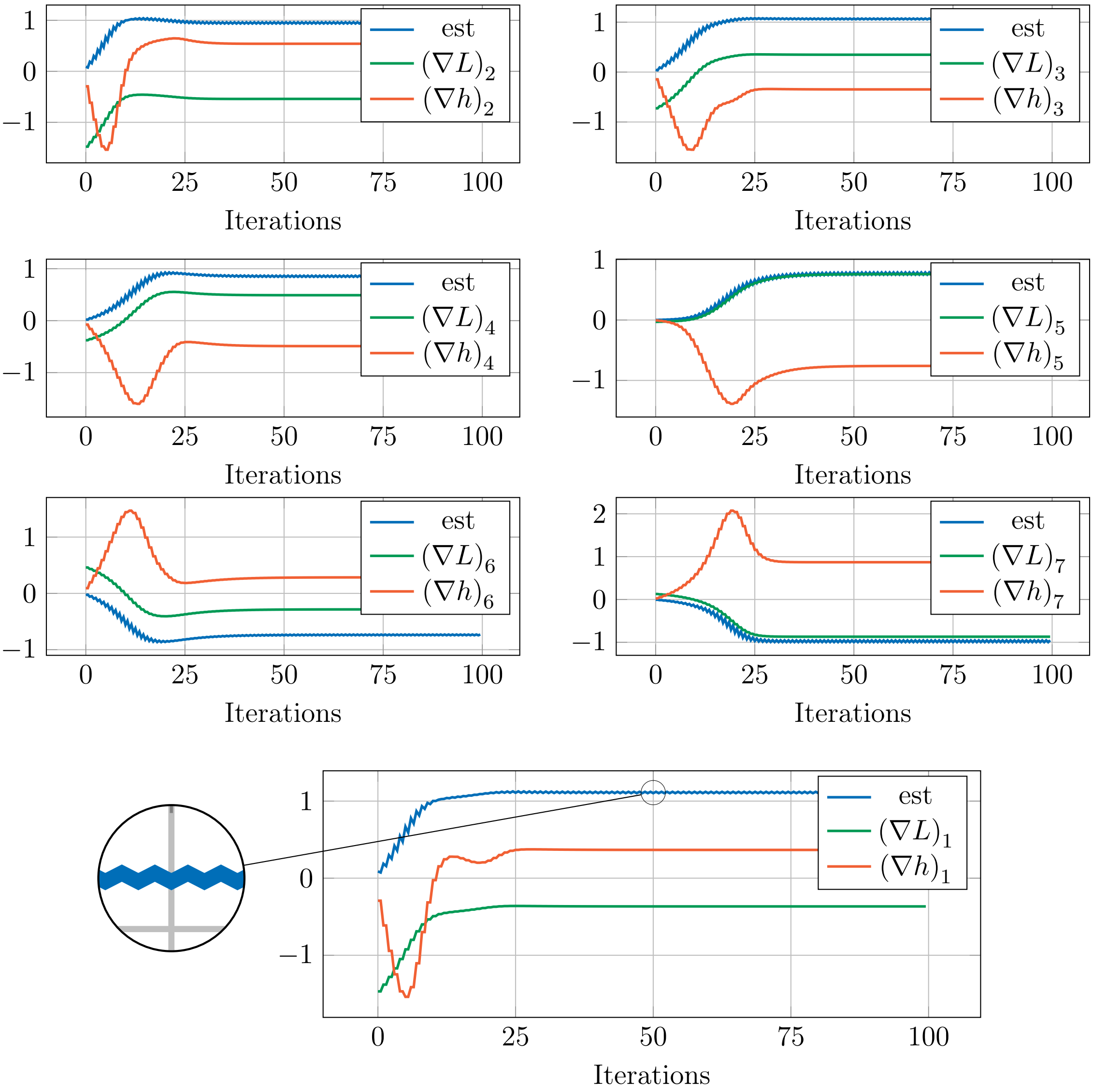
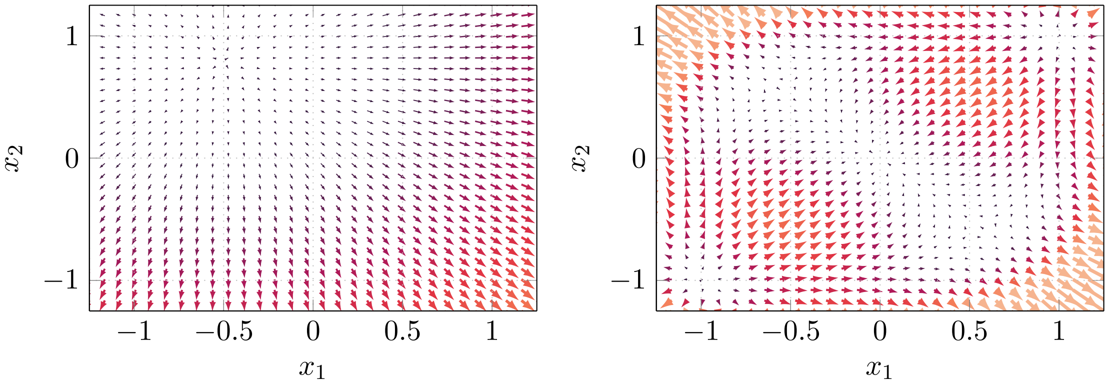


Figure 3.6: Visualization of a single decoding operation. BCH(7, 4) code

It is evident that in all cases, past a certain number of iterations, the estimate starts to oscillate around a particular value. Jointly, the two gradients stop further approaching the value zero. This leads to the two terms of the objective function and in particular the code-constraint polynomial not being minimized. As such, the constraints are not being satisfied and the estimate is not converging towards a valid codeword.

While figure 3.6 shows only one instance of a decoding task with no statistical significance, it is indicative of the general behavior of the algorithm. This can be justified by looking at the gradients themselves. In figure 3.7 the gradients of the negative log-likelihood and the code-constraint polynomial for a repetition code with $n = 2$ are shown. The two valid codewords of the $n = 2$ repetition code can be recognized in figure 3.7b as $\boldsymbol{c}_1 = \begin{bmatrix} -1 & -1 \end{bmatrix}$ and $\boldsymbol{c}_2 = \begin{bmatrix} 1 & 1 \end{bmatrix}$; these are also the points producing the global minima of the code-constraint polynomial. The gradient of the negative log-likelihood points towards the received codeword as can be seen in figure 3.7a, since assuming AWGN and no other information that is the estimate maximizing the likelihood. Looking at figure 3.7b it also becomes apparent why the value of the parameter γ has to be kept small, as mentioned in section 3.1. Local minima are introduced between the codewords, in the areas in which it is not immediately clear which codeword is the most likely one. Increasing the value of γ results in $h\left(\tilde{\boldsymbol{x}}\right)$ dominating the landscape of the objective function, thereby introducing these local minima into the objective function.



(a) $\nabla L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right)$ for a repetition code with $n = 2$ **(b)** $\nabla h\left(\tilde{\boldsymbol{x}}\right)$ for a repetition code with $n = 2$

Figure 3.7: Gradiensts of the negative log-likelihood and the code-constraint polynomial

It is obvious that walking along the gradients in an alternating fashion will produce a net movement in a certain direction, as long as they have a common component. As soon as this common component is exhausted, they will start pulling the estimate in opposing directions, leading to an oscillation as illustrated in figure 3.6. Consequently, this oscillation is an intrinsic property of the structure of the minimization process of the proximal decoding algorithm, where the two parts of the objective function are minimized in an alternating fashion by use of their gradients.

While the initial net movement is generally directed in the right direction owing to the gradient of the negative log-likelihood, the resulting oscillation may well take place in a segment of space not corresponding to a valid codeword, leading to the aforementioned non-convergence of the algorithm. This also partly explains the difference in decoding performance when looking at the BER and FER, as it would lower the amount of bit errors while still yielding an invalid codeword.

The higher the SNR, the more likely the gradient of the negative log-likelihood is to point to a valid codeword. The common component of the two gradients then pulls the estimate closer to a valid codeword before the oscillation takes place. This explains why the decoding performance is significantly better for higher SNRs.

When considering codes with larger n the behavior generally stays the same, with some minor differences. In figure 3.8 the decoding process is visualized for one component of a code with $n = 204$, for a single decoding. The two gradients still eventually oppose each other and the estimate still starts to oscillate, the same as illustrated in figure 3.6 based on a code with $n = 7$. However, in this case, the gradient of the code-constraint polynomial itself starts to oscillate, its average value being such that the effect of the gradient of the negative log-likelihood is counteracted.

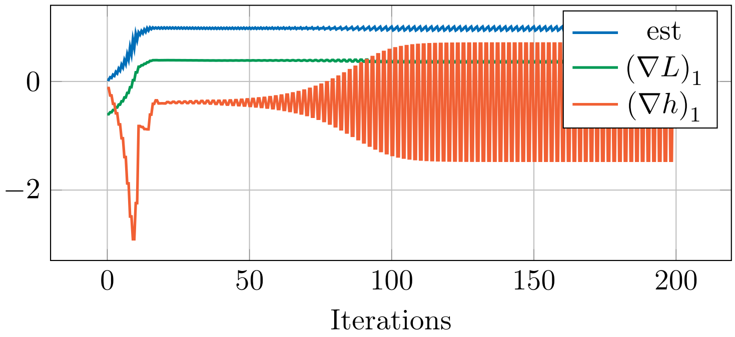


Figure 3.8: Visualization of a single decoding operation. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

In conclusion, as a general rule, the proximal decoding algorithm reaches an oscillatory state which it cannot escape as a consequence of its structure. In this state the constraints may not be satisfied, leading to the algorithm exhausting its maximum number of iterations without converging and returning an invalid codeword.

3.2.4 Computational Performance

In order to determine the time complexity of proximal decoding for an AWGN channel, the decoding process as depicted in algorithm 3.1 is considered.

When dealing with LDPC codes, i.e., the number of non-zero entries in the parity-check matrix $\boldsymbol{H}$ grows linearly with n, the two recursive steps in lines 3 and 4 have time complexity $\mathcal{O}(n)$ [WT22, Sec. 4.1]. The sign operation in line 5 also has $\mathcal{O}(n)$ time complexity, as it only depends on the n components of $\boldsymbol{s}$. Given the sparsity of the matrix $\boldsymbol{H}$, evaluating the parity-check condition has linear time complexity as well, since at most n additions and multiplications have to be performed. This means that the overall time complexity of proximal decoding for LDPC codes in an AWGN channel is $\mathcal{O}(n)$, which is practical since it is the same as that of BP.

This theoretical analysis is also corroborated by the practical results shown in figure 3.9. The codes considered are the BCH(31, 11) and BCH(31, 26) codes, several (3, 6) regular LDPC codes ([Mac23, 96.3.965, 204.33.484, 408.33.844]), a (5,10) regular LDPC code ([Mac23, 204.55.187]) and a progressive edge growth construction code ([Mac23, PEGReg252x504]). Some deviations from linear behavior are unavoidable, since not all codes considered are actually LDPC codes, or LDPC codes constructed according to the same scheme. Nonetheless, a generally linear relationship between the average time needed to decode a received frame and the length n of the frame can be observed. These results were generated on an Intel Core i7-7700HQ 4-core CPU, running at 2.80 GHz and utilizing all cores.

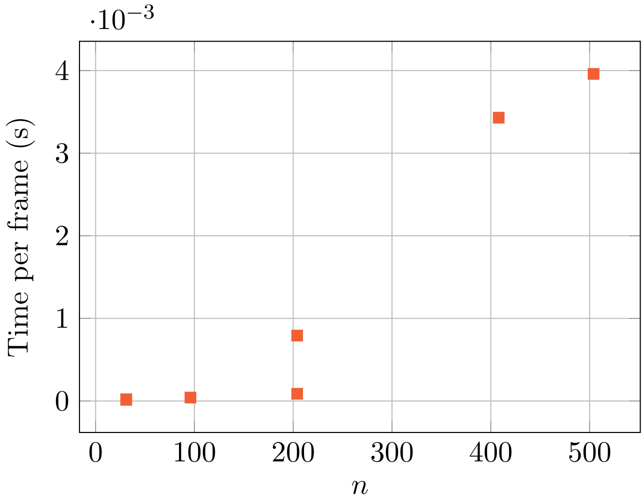


Figure 3.9: Timing requirements of the proximal decoding implementation

3.3 Improved Implementation

As mentioned earlier, frame errors seem to mainly stem from decoding failures. Coupled with the fact that the BER indicates significantly better performance than the FER, this leads to the assumption that only a small number of components of the estimated vector may be responsible for an invalid result. If it was possible to limit the number of possibly wrong components of the estimate to a small subset, an ML-decoding step could be performed on a limited number of possible results ("ML-in-the-list", as it is called) to improve the decoding performance. This concept is pursued in this section.

First, a guideline must be found with which to evaluate the probability that a given component of an estimate is wrong. One compelling observation is that the closer an estimate is to being at a valid codeword, the smaller the magnitude of the gradient of the code-constraint polynomial, as illustrated in figure 3.7. This gives rise to the notion that some property or behavior of $\nabla h\left(\tilde{\boldsymbol{x}}\right)$ may be related in its magnitude to the confidence that a given bit is correct. And indeed, the magnitude of the oscillation of $\nabla h\left(\tilde{\boldsymbol{x}}\right)$ (introduced previously in section 3.2.3 and shown in figure 3.8) and the probability of having a bit error are strongly correlated, a relationship being depicted in figure 3.10. The x-axis depicts the variance in $\nabla h\left(\tilde{\boldsymbol{x}}\right)$ after the 100th iteration, and the y-axis depicts whether there is a bit error. While this is not exactly the magnitude of the oscillation, it is proportional and easier to compute. The datapoints are taken from a single decoding operation.

Using this observation as a rule to determine the $N \in \mathbb{N}$ most probably wrong bits, all variations of the estimate with those bits modified can be generated. An ML-in-the-list step can then be performed to determine the most likely candidate. This process is outlined in algorithm 3.2. Its only difference to algorithm 3.1 is that instead of returning the last estimate when no valid result is reached, an ML-in-the-list step is performed.

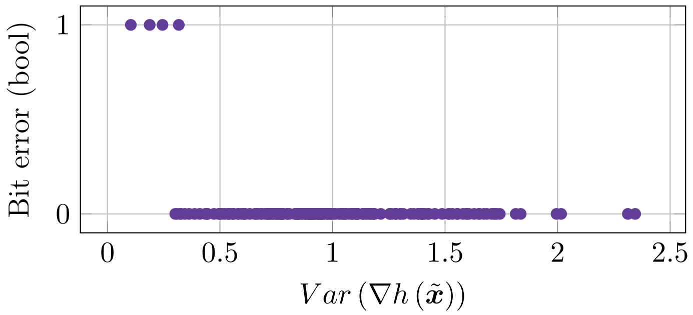


Figure 3.10: Correlation between the occurrence of a bit error and the amplitude of oscillation of the gradient of the code-constraint polynomial. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

```
$\boldsymbol{s} \leftarrow \boldsymbol{0}$
for $K$ iterations do
    $\boldsymbol{r} \leftarrow \boldsymbol{s} - \omega \nabla L\left(\boldsymbol{y} \mid \boldsymbol{s}\right)$
    $\boldsymbol{s} \leftarrow \boldsymbol{r} - \gamma \nabla h\left(\boldsymbol{r}\right)$
    $\hat{\boldsymbol{x}} \leftarrow \text{sign}\left(\boldsymbol{s}\right)$
    if $\boldsymbol{H}\hat{\boldsymbol{c}} = \boldsymbol{0}$
        return $\hat{\boldsymbol{c}}$
    end if
end for
Find $N$ most probably wrong bits
Generate variations $\hat{\boldsymbol{c}}_l$, $l = 1, \ldots, 2^N$ of $\hat{\boldsymbol{c}}$ with the $N$ bits modified
Compute $d_H\left(\hat{\boldsymbol{c}}_l, \hat{\boldsymbol{c}}\right)$ for all valid codewords $\hat{\boldsymbol{c}}_l$
Output $\hat{\boldsymbol{c}}_l$ with lowest $d_H\left(\hat{\boldsymbol{c}}_l, \hat{\boldsymbol{c}}\right)$
```

Algorithm 3.2: Improved proximal decoding algorithm

Figure 3.11 shows the gain that can be achieved when the number N is chosen to be 12. Again, three values of gamma are chosen, for which the BER, FER and decoding failure rate are plotted. The simulation results for the original proximal decoding algorithm are shown with solid lines and the results for the improved version are shown with dashed lines. For the case of $\gamma = 0.05$, the number of frame errors produced for the datapoints at 6 dB, 6.5 dB and 7 dB are 70, 17 and 2, respectively. The gain seems to depend on the value of γ, as well as becoming more pronounced for higher SNR values. This is to be expected, since with higher SNR values the number of bit errors decreases, making the correction of those errors in the ML-in-the-list step more likely. In figure 3.15 the decoding performance between proximal decoding and the improved algorithm is compared for a number of different codes. In the case of the code with $n = 504$ shown in figure 3.15f, only 76 frame errors were produced to generate the point for the improved algorithm for $\gamma = 0.05$ at 5.5 dB. Similar behavior can be observed in all cases, with varying improvement over standard proximal decoding. In some cases, a gain of up to 1 dB or higher can be achieved.

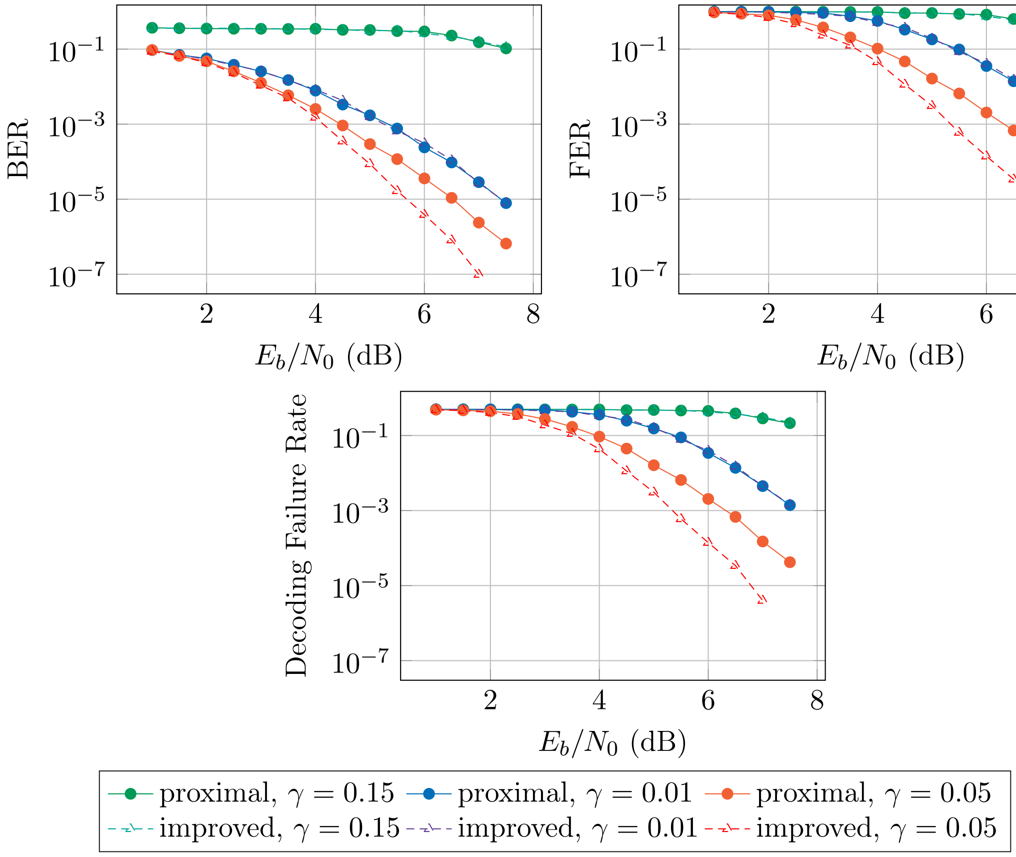


Figure 3.11: Comparison of the decoding performance between proximal decoding and the improved implementation. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

The average run time of the improved algorithm depends on the parameter N. The time complexity, however, does not: it is still linear with respect to n. Interestingly, the improved algorithm does not have a much different average run time than proximal decoding, because the computationally expensive ML-in-the-list step is only performed when the proximal decoding algorithm produces an invalid result, which in absolute terms happens relatively infrequently. This is illustrated in figure 3.12, where the average time needed to decode a single received frame is visualized for proximal decoding as well as for the improved algorithm. The same codes as before are considered, i.e., the BCH(31, 11) and BCH(31, 26) codes, several (3, 6) regular LDPC codes ([Mac23, 96.3.965, 204.33.484, 408.33.844]), a (5,10) regular LDPC code ([Mac23, 204.55.187]) and a progressive edge growth construction code ([Mac23, PEGReg252x504]). It should be noted that some variability in the data is to be expected, since the timing of the actual simulations depends on a multitude of other parameters such as the scheduling choices of the operating system as well as variations in the implementations themselves. Nevertheless, the empirical data, at least in part, supports the theoretical considerations.

In conclusion, the decoding performance of proximal decoding can be improved by appending an ML-in-the-list step when the algorithm does not produce a valid result. The gain can in some cases be as high as 1 dB and is achievable with negligible computational performance penalty. The improvement is mainly noticeable for higher SNR values and depends on the code as well as the chosen parameters.

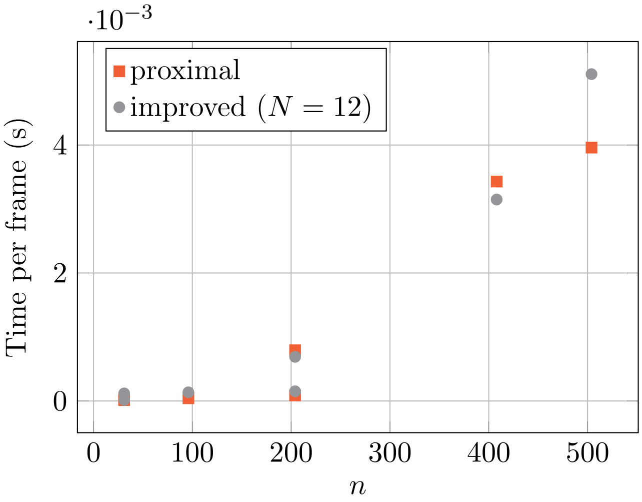
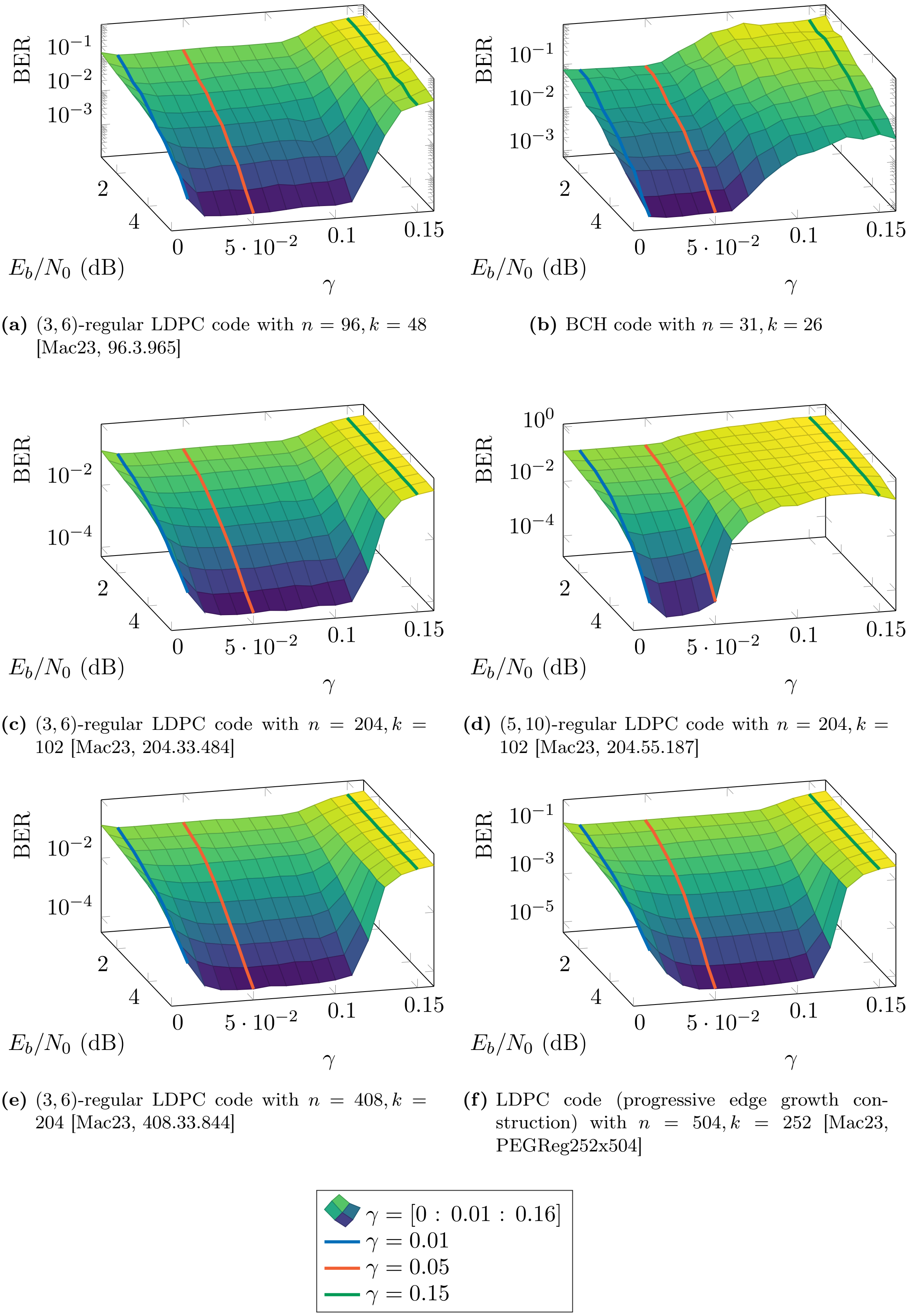


Figure 3.12: Comparison of the timing requirements of the implementations of proximal decoding and the improved algorithm



(a) (3, 6)-regular LDPC code with $n = 96, k = 48$ [Mac23, 96.3.965]

(b) BCH code with $n = 31, k = 26$

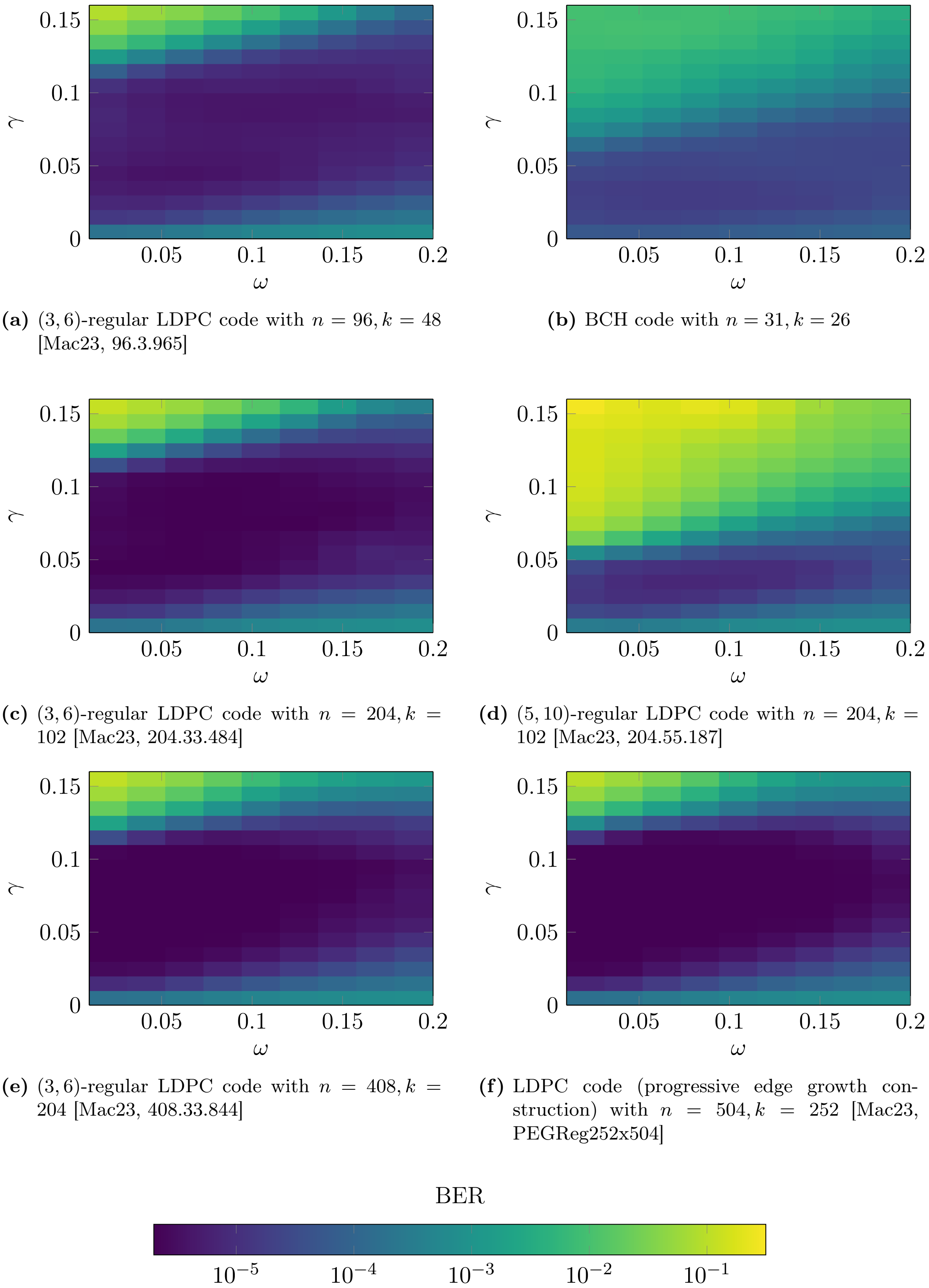
(c) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(d) (5, 10)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.55.187]

(e) (3, 6)-regular LDPC code with $n = 408, k = 204$ [Mac23, 408.33.844]

(f) LDPC code (progressive edge growth construction) with $n = 504, k = 252$ [Mac23, PEGReg252x504]

Figure 3.13: Dependence of the BER on the value of the parameter γ for various codes



(a) (3, 6)-regular LDPC code with $n = 96, k = 48$ [Mac23, 96.3.965]

(b) BCH code with $n = 31, k = 26$

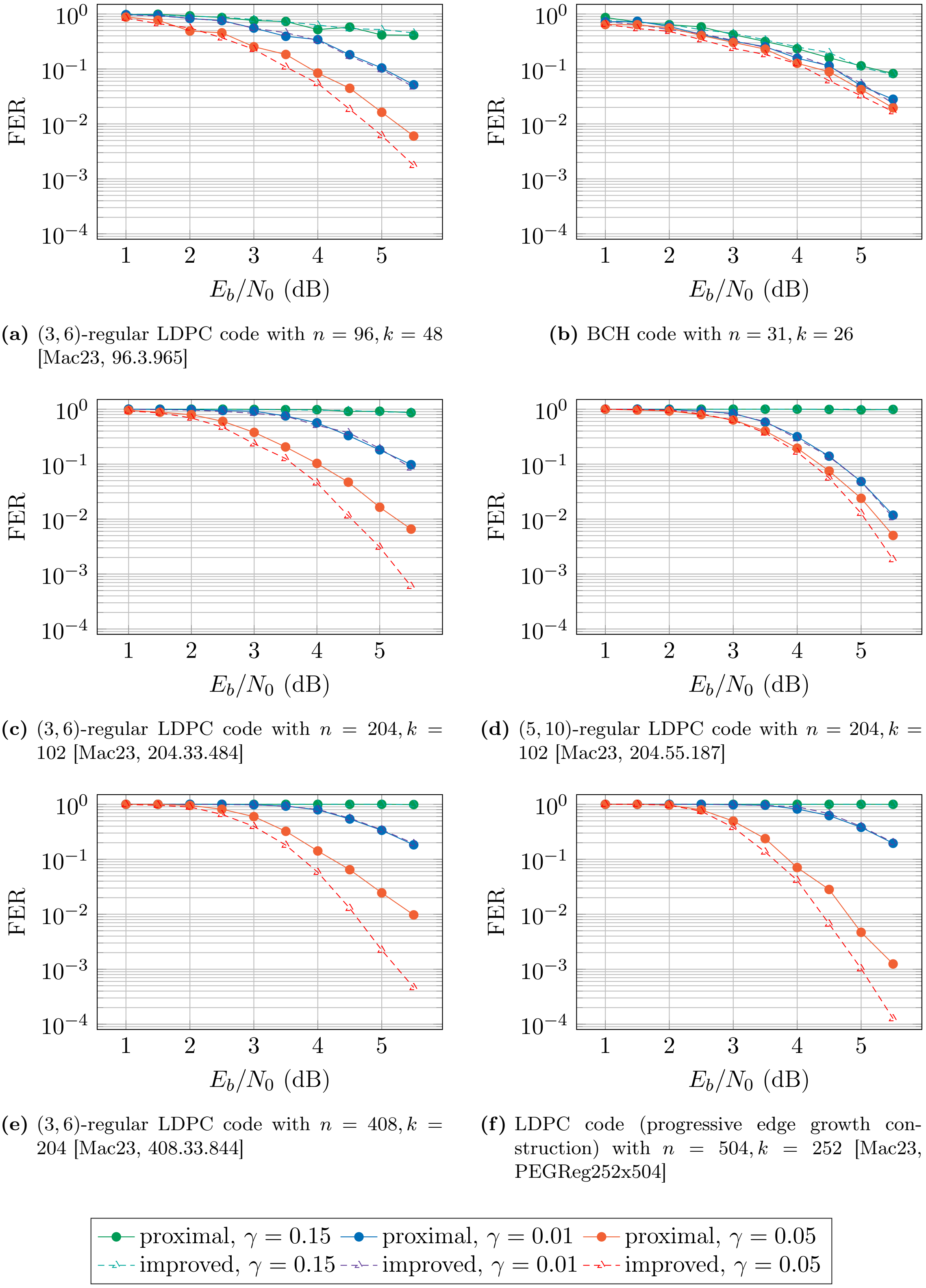
(c) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(d) (5, 10)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.55.187]

(e) (3, 6)-regular LDPC code with $n = 408, k = 204$ [Mac23, 408.33.844]

(f) LDPC code (progressive edge growth construction) with $n = 504, k = 252$ [Mac23, PEGReg252x504]

Figure 3.14: The BER as a function of γ and ω for various codes



(a) (3, 6)-regular LDPC code with $n = 96, k = 48$ [Mac23, 96.3.965]

(b) BCH code with $n = 31, k = 26$

(c) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(d) (5, 10)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.55.187]

(e) (3, 6)-regular LDPC code with $n = 408, k = 204$ [Mac23, 408.33.844]

(f) LDPC code (progressive edge growth construction) with $n = 504, k = 252$ [Mac23, PEGReg252x504]

Figure 3.15: Comparison of the decoding performance of proximal decoding and the improved algorithm for various codes

4 LP Decoding using ADMM

This chapter is concerned with LP decoding - the reformulation of the decoding problem as a linear program. More specifically, the LP decoding problem is solved using ADMM. First, the general field of LP decoding is introduced. The application of ADMM to the decoding problem is explained and some notable implementation details are mentioned. Finally, the behavior of the algorithm is examined based on simulation results.

4.1 LP Decoding

LP decoding is a subject area introduced by Feldman et al. [FWK05]. They reframe the decoding problem as an *integer linear program* and subsequently present two relaxations into *linear programs*, one representing an LP formulation of exact ML decoding and one, which is an approximation with a more manageable representation. To solve the resulting linear program, various optimization methods can be used (see for example [TS06], [Von08], [ZS13], [Zha+19]).

Feldman et al. begin by looking at the ML decoding problem[1]

$$\hat{\boldsymbol{c}}_{\text{ML}} = \underset{\boldsymbol{c}\in\mathcal{C}}{\operatorname{argmax}}\, f_{\boldsymbol{Y}|\boldsymbol{C}}\left(\boldsymbol{y} \mid \boldsymbol{c}\right). \tag{4.1}$$

Assuming a memoryless channel, equation (4.1) can be rewritten in terms of the *log-likelihood ratio*s (LLRs) γ_i [Fel03, Sec. 2.5]:

$$\hat{\boldsymbol{c}}_{\text{ML}} = \underset{\boldsymbol{c}\in\mathcal{C}}{\operatorname{argmin}} \sum_{i=1}^{n} \gamma_i c_i, \qquad \gamma_i = \ln\left(\frac{f_{Y_i|C_i}\left(y_i \mid c_i = 0\right)}{f_{Y_i|C_i}\left(y_i \mid c_i = 1\right)}\right).$$

The authors propose using the following cost function[2] for the LP decoding problem:

$$g\left(\boldsymbol{c}\right) = \sum_{i=1}^{n} \gamma_i c_i = \boldsymbol{\gamma}^{\text{T}}\boldsymbol{c}.$$

With this cost function, the exact integer linear program formulation of ML decoding becomes

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\gamma}^{\text{T}}\boldsymbol{c} \\ \text{subject to} \quad & \boldsymbol{c} \in \mathcal{C}. \end{aligned}$$

As solving integer linear programs is generally NP-hard, this decoding problem has to be approximated by a problem with looser constraints. A technique called *relaxation* is applied: relaxing the constraints, thereby broadening the considered domain (e.g., by lifting

[1] They assume that all codewords are equally likely to be transmitted, making the ML and MAP decoding problems equivalent.

[2] In this context, *cost function* and *objective function* have the same meaning.

the integer requirement). First, the authors present an equivalent LP formulation of exact ML decoding, redefining the constraints in terms of the codeword polytope

$$\text{poly}\left(\mathcal{C}\right) = \left\{ \sum_{\boldsymbol{c} \in \mathcal{C}} \alpha_{\boldsymbol{c}} \boldsymbol{c} : \alpha_{\boldsymbol{c}} \geq 0, \sum_{\boldsymbol{c} \in \mathcal{C}} \alpha_{\boldsymbol{c}} = 1 \right\},$$

which represents the *convex hull* of all possible codewords, i.e., the convex set of linear combinations of all codewords. This corresponds to simply lifting the integer requirement. However, since the number of constraints needed to characterize the codeword polytope is exponential in the code length, this formulation is relaxed further. By observing that each check node defines its own local single parity-check code, and, thus, its own *local codeword polytope*, the *relaxed codeword polytope* $\overline{Q}$ is defined as the intersection of all local codeword polytopes. This consideration leads to constraints that can be described as follows [ZS13, Sec. II, A]:

$$\boldsymbol{T}_j \tilde{\boldsymbol{c}} \in \mathcal{P}_{d_j} \qquad \forall j \in \mathcal{J},$$

where $\mathcal{P}_{d_j}$ is the *check polytope*, i.e., the convex hull of all binary vectors of length d_j with even parity[3], and $\boldsymbol{T}_j$ is the *transfer matrix*, which selects the neighboring variable nodes of check node j (i.e., the relevant components of $\tilde{\boldsymbol{c}}$ for parity-check j). For example, if the jth row of the parity-check matrix $\boldsymbol{H}$ was $\boldsymbol{h}_j = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$, the transfer matrix would be [ZS13, Sec. II, A]

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

In figure 4.1, the two relaxations are compared for an examplary code, which is described by the generator and parity-check matrices

$$\boldsymbol{G} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \tag{4.2}$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.3}$$

and has only two possible codewords:

$$\mathcal{C} = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Figure 4.1a shows the domain of exact ML decoding. The first relaxation onto the codeword polytope poly $(\mathcal{C})$ is shown in figure 4.1b; this expresses the constraints for the equivalent linear program to exact ML decoding. poly $(\mathcal{C})$ is further relaxed onto the relaxed codeword polytope $\overline{Q}$, shown in figure 4.1d. Figure 4.1c shows how $\overline{Q}$ is formed by intersecting the local codeword polytopes of each check node.

[3] Essentially $\mathcal{P}_{d_j}$ is the set of vectors that satisfy parity-check j, but extended to the continuous domain.

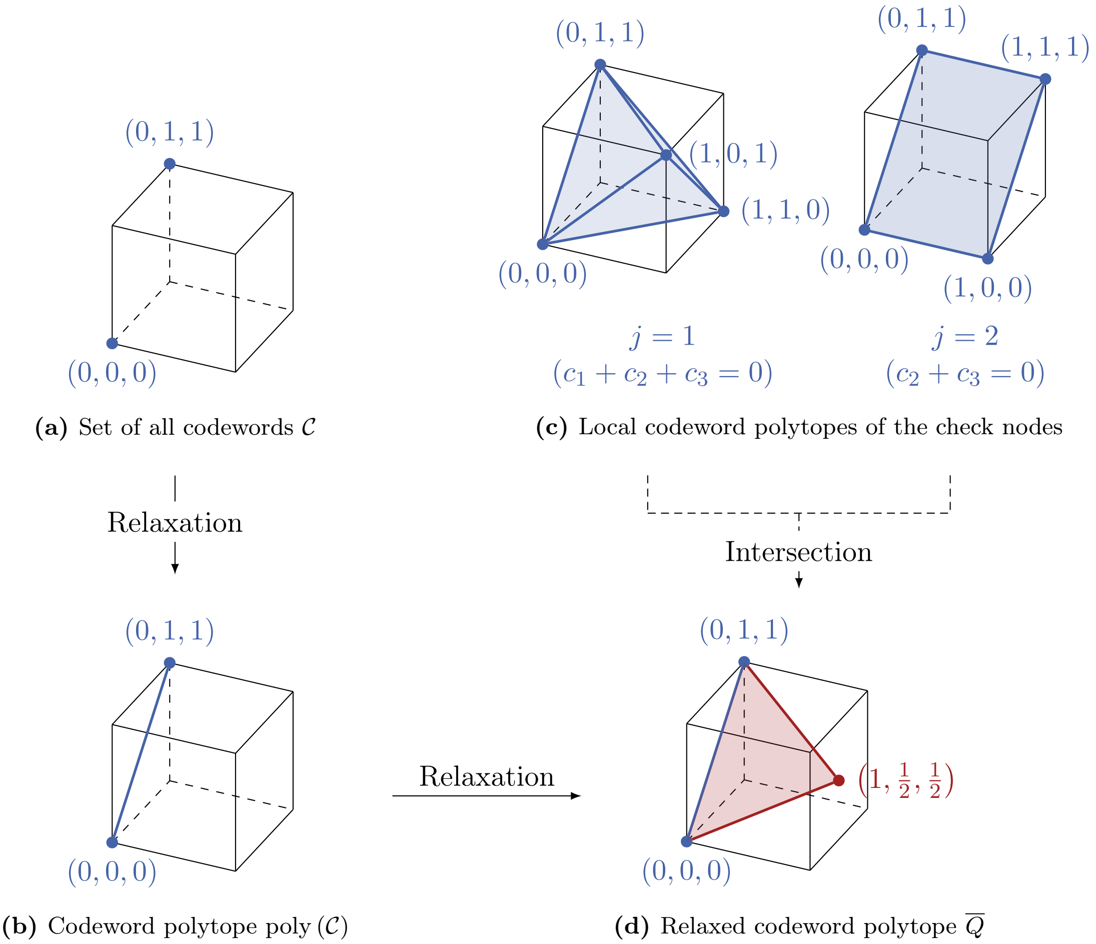


Figure 4.1: Visualization of the codeword polytope and the relaxed codeword polytope of the code described by equations (4.2) and (4.3)

It can be seen that the relaxed codeword polytope $\overline{Q}$ introduces vertices with fractional values; these represent erroneous non-codeword solutions to the linear program and correspond to the so-called *pseudo-codewords* introduced in [FWK05]. However, since for LDPC codes $\overline{Q}$ scales linearly with n instead of exponentially, it is a lot more tractable for practical applications.

The resulting formulation of the relaxed optimization problem becomes

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\gamma}^{\mathrm{T}} \tilde{\boldsymbol{c}} \\ \text{subject to} \quad & \boldsymbol{T}_j \tilde{\boldsymbol{c}} \in \mathcal{P}_{d_j} \quad \forall j \in \mathcal{J}. \end{aligned} \tag{4.4}$$

One aspect making LP decoding especially appealing is the very strong theoretical guarantee that comes with it, called the *ML certificate property* [FWK05, Sec. III. B.]. This is the property that when a valid result is produced by an LP decoder, it is always the ML codeword. This leads to an interesting application of LP decoding to approximate ML decoding behavior, by successively adding redundant parity-checks until a valid result is returned [TS06, Sec. IV.].

4.2 Decoding Algorithm and Implementation

The LP decoding formulation in section 4.1 is a very general one that can be solved with a number of different optimization methods. In this work ADMM is examined, as its distributed nature allows for a very efficient implementation. LP decoding using ADMM can be regarded as a message passing algorithm with separate variable- and check-node update steps; the resulting algorithm has a striking similarity to BP and its computational complexity has been demonstrated to compare favorably to BP [Bar+13], [ZS13].

The LP decoding problem in (4.4) can be slightly rewritten using the auxiliary variables $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m$:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\gamma}^{\mathrm{T}} \tilde{\boldsymbol{c}} \\ \text{subject to} \quad & \left.\begin{aligned} \boldsymbol{T}_j \tilde{\boldsymbol{c}} &= \boldsymbol{z}_j \\ \boldsymbol{z}_j &\in \mathcal{P}_{d_j} \end{aligned}\right\} \quad \forall j \in \mathcal{J}. \end{aligned} \tag{4.5}$$

In this form, the problem almost fits the ADMM template described in section 2.6, except for the fact that there are multiple equality constraints $\boldsymbol{T}_j \tilde{\boldsymbol{c}} = \boldsymbol{z}_j$ and the additional constraints $\boldsymbol{z}_j \in \mathcal{P}_{d_j} \, \forall \, j \in \mathcal{J}$. The multiple constraints can be addressed by introducing additional terms in the augmented lagrangian:

$$\mathcal{L}_\mu \left(\tilde{\boldsymbol{c}}, (\boldsymbol{z})_{j=1}^m, (\boldsymbol{\lambda})_{j=1}^m \right) = \boldsymbol{\gamma}^{\mathrm{T}} \tilde{\boldsymbol{c}} + \sum_{j \in \mathcal{J}} \boldsymbol{\lambda}_j^{\mathrm{T}} \left(\boldsymbol{T}_j \tilde{\boldsymbol{c}} - \boldsymbol{z}_j \right) + \frac{\mu}{2} \sum_{j \in \mathcal{J}} \lVert \boldsymbol{T}_j \tilde{\boldsymbol{c}} - \boldsymbol{z}_j \rVert_2^2.$$

The additional constraints remain in the dual optimization problem:

$$\text{maximize} \quad \min_{\substack{\tilde{\boldsymbol{c}} \\ \boldsymbol{z}_j \in \mathcal{P}_{d_j} \forall \, j \in \mathcal{J}}} \mathcal{L}_\mu \left(\tilde{\boldsymbol{c}}, (\boldsymbol{z})_{j=1}^m, (\boldsymbol{\lambda})_{j=1}^m \right).$$

The steps to solve the dual problem then become:

$$\begin{aligned} \tilde{\boldsymbol{c}} &\leftarrow \operatorname*{argmin}_{\tilde{\boldsymbol{c}}} \mathcal{L}_\mu \left(\tilde{\boldsymbol{c}}, (\boldsymbol{z})_{j=1}^m, (\boldsymbol{\lambda})_{j=1}^m \right) \\ \boldsymbol{z}_j &\leftarrow \operatorname*{argmin}_{\boldsymbol{z}_j \in \mathcal{P}_{d_j}} \mathcal{L}_\mu \left(\tilde{\boldsymbol{c}}, (\boldsymbol{z})_{j=1}^m, (\boldsymbol{\lambda})_{j=1}^m \right) && \forall j \in \mathcal{J} \\ \boldsymbol{\lambda}_j &\leftarrow \boldsymbol{\lambda}_j + \mu \left(\boldsymbol{T}_j \tilde{\boldsymbol{c}} - \boldsymbol{z}_j \right) && \forall j \in \mathcal{J}. \end{aligned}$$

Luckily, the additional constraints only affect the $\boldsymbol{z}_j$-update steps. Furthermore, the $\boldsymbol{z}_j$-update steps can be shown to be equivalent to projections onto the check polytopes $\mathcal{P}_{d_j}$

and the $\tilde{\boldsymbol{c}}$-update can be computed analytically[4] [Bar+13, Sec. III. B.]:

$$
\begin{aligned}
\tilde{c}_i &\leftarrow \frac{1}{d_i}\left(\sum_{j\in N_v(i)}\Big(\left(\boldsymbol{z}_j\right)_i - \frac{1}{\mu}\left(\boldsymbol{\lambda}_j\right)_i\Big) - \frac{\gamma_i}{\mu}\right) && \forall i \in \mathcal{I}\\
\boldsymbol{z}_j &\leftarrow \Pi_{\mathcal{P}_{d_j}}\left(\boldsymbol{T}_j\tilde{\boldsymbol{c}} + \frac{\boldsymbol{\lambda}_j}{\mu}\right) && \forall j \in \mathcal{J}\\
\boldsymbol{\lambda}_j &\leftarrow \boldsymbol{\lambda}_j + \mu\left(\boldsymbol{T}_j\tilde{\boldsymbol{c}} - \boldsymbol{z}_j\right) && \forall j \in \mathcal{J}.
\end{aligned}
$$

It should be noted that all of the $\boldsymbol{z}_j$-updates can be computed simultaneously, as they are independent of one another. The same is true for the updates of the individual components of $\tilde{\boldsymbol{c}}$. This representation can be slightly simplified by substituting $\boldsymbol{\lambda}_j = \mu \cdot \boldsymbol{u}_j \, \forall \, j \in \mathcal{J}$:

$$
\tilde{c}_i \leftarrow \frac{1}{d_i}\left(\sum_{j\in N_v(i)}\Big(\left(\boldsymbol{z}_j\right)_i - \left(\boldsymbol{u}_j\right)_i\Big) - \frac{\gamma_i}{\mu}\right) \quad \forall i \in \mathcal{I} \tag{4.6}
$$

$$
\boldsymbol{z}_j \leftarrow \Pi_{\mathcal{P}_{d_j}}\left(\boldsymbol{T}_j\tilde{\boldsymbol{c}} + \boldsymbol{u}_j\right) \quad \forall j \in \mathcal{J} \tag{4.7}
$$

$$
\boldsymbol{u}_j \leftarrow \boldsymbol{u}_j + \boldsymbol{T}_j\tilde{\boldsymbol{c}} - \boldsymbol{z}_j \quad \forall j \in \mathcal{J}. \tag{4.8}
$$

The reason ADMM is able to perform so well is due to the relocation of the constraints $\boldsymbol{T}_j\tilde{\boldsymbol{c}}_j \in \mathcal{P}_{d_j} \, \forall \, j \in \mathcal{J}$ into the objective function itself. The minimization of the new objective function can then take place simultaneously with respect to all $\boldsymbol{z}_j, j \in \mathcal{J}$. Effectively, all of the $|\mathcal{J}|$ parity constraints can be handled at the same time. This can also be understood by interpreting the decoding process as a message-passing algorithm [Bar+13, Sec. III. D.], [ZS13, Sec. II. B.], depicted in algorithm 4.1.

```
Initialize $\tilde{\boldsymbol{c}}, \boldsymbol{z}_{[1:m]}$ and $\boldsymbol{u}_{[1:m]}$
while $\sum_{j\in\mathcal{J}} \lVert \boldsymbol{T}_j\tilde{\boldsymbol{c}} - \boldsymbol{z}_j \rVert_2 \ge \epsilon_{\text{pri}}$ or $\sum_{j\in\mathcal{J}} \lVert \boldsymbol{z}'_j - \boldsymbol{z}_j \rVert_2 \ge \epsilon_{\text{dual}}$ do
    for $j$ in $\mathcal{J}$ do
        $\boldsymbol{z}_j \leftarrow \Pi_{\mathcal{P}_{d_j}}\left(\boldsymbol{T}_j\tilde{\boldsymbol{c}} + \boldsymbol{u}_j\right)$
        $\boldsymbol{u}_j \leftarrow \boldsymbol{u}_j + \boldsymbol{T}_j\tilde{\boldsymbol{c}} - \boldsymbol{z}_j$
    end for
    for $i$ in $\mathcal{I}$ do
        $\tilde{c}_i \leftarrow \frac{1}{d_i}\left(\sum_{j\in N_v(i)}\Big(\left(\boldsymbol{z}_j\right)_i - \left(\boldsymbol{u}_j\right)_i\Big) - \frac{\gamma_i}{\mu}\right)$
    end for
end while
return $\tilde{\boldsymbol{c}}$
```

Algorithm 4.1: LP decoding using ADMM interpreted as a message passing algorithm[5]

[4] In the $\tilde{c}_i$-update rule, the term $\left(\boldsymbol{z}_j\right)_i$ is a slight abuse of notation, as $\boldsymbol{z}_j$ has less components than there are variable-nodes i. What is actually meant is the component of $\boldsymbol{z}_j$ that is associated with the variable node i, i.e., $\left(\boldsymbol{T}_j^{\mathrm{T}}\boldsymbol{z}_j\right)_i$. The same is true for $\left(\boldsymbol{\lambda}_j\right)_i$.

[5] $\epsilon_{\text{pri}} > 0$ and $\epsilon_{\text{dual}} > 0$ are additional parameters defining the tolerances for the stopping criteria of the algorithm. The variable $\boldsymbol{z}'_j$ denotes the value of $\boldsymbol{z}_j$ in the previous iteration.

The $\boldsymbol{z}_j$- and $\boldsymbol{\lambda}_j$-updates can be understood as a check-node update step (lines 3-6) and the $\tilde{c}_i$-updates can be understood as a variable-node update step (lines 7-9 in figure 4.1). The updates for each variable- and check-node can be perfomed in parallel.

A technique called *over-relaxation* can be employed to further improve convergence, introducing the over-relaxation parameter ρ. This consists of computing the term $\rho \boldsymbol{T}_j \tilde{\boldsymbol{c}} - (1-\rho)\, \boldsymbol{z}_j$ before the $\boldsymbol{z}_j$ and $\boldsymbol{u}_j$ update steps (lines 4 and 5 of algorithm 4.1) and subsequently replacing $\boldsymbol{T}_j \tilde{\boldsymbol{c}}$ with the computed value in the two updates [Boy+11, Sec. 3.4.3].

The main computational effort in solving the linear program amounts to computing the projection operation $\Pi_{\mathcal{P}_{d_j}}(\cdot)$ onto each check polytope. Various different methods to perform this projection have been proposed (e.g., in [Bar+13], [ZS13], [Gen+20]). The method chosen here is the one presented in [Bar+13].

The development process used to implement this decoding algorithm was the same as outlined in section 3.1 for proximal decoding. First, an initial version was implemented in Python, before repeating the process using C++ to achieve higher performance. Again, the performance can be increased by reframing the operations in such a way that the computation can take place primarily with element-wise operations and matrix-vector multiplication, since these operations are highly optimized in the software libraries used for the implementation.

In the summation operation in line 8 of algorithm 4.1, the components of each $\boldsymbol{z}_j$ and $\boldsymbol{u}_j$ relating to a given VN i have to be found. This operation can be streamlined by observing that the transfer matrices $\boldsymbol{T}_j,\ j \in \mathcal{J}$ are able to perform the mapping they were devised for in both directions: with $\boldsymbol{T}_j \tilde{\boldsymbol{c}}$, the d_j components of $\tilde{\boldsymbol{c}}$ required for parity check i are selected; with $\boldsymbol{T}_j^{\mathrm{T}} \boldsymbol{z}_j$, the d_j components of $\boldsymbol{z}_j$ can be mapped onto a vector of length n, each component at the position corresponding to the VN it relates to. Using this observation, the sum can be written as

$$\sum_{j \in N_v(i)} \left(\boldsymbol{T}_j^{\mathrm{T}} \left(\boldsymbol{z}_j - \boldsymbol{u}_j \right) \right)_i .$$

Further noticing that the vectors $\boldsymbol{T}_j^{\mathrm{T}} \left(\boldsymbol{z}_j - \boldsymbol{u}_j \right)$ unrelated to VN i have 0 as the ith component, the set of indices the summation takes place over can be extended to $\mathcal{J}$, allowing the expression to be rewritten as

$$\sum_{j \in \mathcal{J}} \left(\boldsymbol{T}_j^{\mathrm{T}} \left(\boldsymbol{z}_j - \boldsymbol{u}_j \right) \right)_i = \left(\sum_{j \in \mathcal{J}} \boldsymbol{T}_j^{\mathrm{T}} \left(\boldsymbol{z}_j - \boldsymbol{u}_j \right) \right)_i .$$

Defining[6]

$$\boldsymbol{d} := \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{s} := \sum_{j \in \mathcal{J}} \boldsymbol{T}_j^{\mathrm{T}} \left(\boldsymbol{z}_j - \boldsymbol{u}_j \right),$$

the $\tilde{\boldsymbol{c}}$ update can then be rewritten as

$$\tilde{\boldsymbol{c}} \leftarrow \boldsymbol{d}^{\circ(-1)} \circ \left(\boldsymbol{s} - \frac{1}{\mu} \gamma \right) .$$

[6] In this case $d_1, \ldots, d_n$ refer to the degree of the variable nodes, i.e., $d_i,\ i \in \mathcal{I}$.

This modified version of the decoding process is depicted in algorithm 4.2.

```
Initialize $\tilde{\boldsymbol{c}}, \boldsymbol{z}_{[1:m]}$ and $\boldsymbol{u}_{[1:m]}$
while $\sum_{j\in\mathcal{J}} \lVert \boldsymbol{T}_j\tilde{\boldsymbol{c}} - \boldsymbol{z}_j \rVert_2 \geq \epsilon_{\text{pri}}$ or $\sum_{j\in\mathcal{J}} \lVert \boldsymbol{z}'_j - \boldsymbol{z}_j \rVert_2 \geq \epsilon_{\text{dual}}$ do
    $\boldsymbol{s} \leftarrow \boldsymbol{0}$
    for $j$ in $\mathcal{J}$ do
        $\boldsymbol{z}_j \leftarrow \Pi_{\mathcal{P}_{d_j}}\left(\boldsymbol{T}_j\tilde{\boldsymbol{c}} + \boldsymbol{u}_j\right)$
        $\boldsymbol{u}_j \leftarrow \boldsymbol{u}_j + \boldsymbol{T}_j\tilde{\boldsymbol{c}} - \boldsymbol{z}_j$
        $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{T}_j\left(\boldsymbol{z}_j - \boldsymbol{u}_j\right)$
    end for
    for $i$ in $\mathcal{I}$ do
        $\tilde{\boldsymbol{c}} \leftarrow \boldsymbol{d}^{\circ(-1)} \circ \left(\boldsymbol{s} - \frac{1}{\mu}\boldsymbol{\gamma}\right)$
    end for
end while
return $\tilde{\boldsymbol{c}}$
```

Algorithm 4.2: The LP decoding using ADMM algorithm with rewritten update steps

4.3 Analysis and Simulation Results

In this section, LP decoding using ADMM is examined based on simulation results for various codes. First, the effect of the different parameters and how their values should be chosen is investigated. Subsequently, the decoding performance is observed and compared to that of BP. Finally, the computational performance of the implementation and time complexity of the algorithm are studied.

As was the case in chapter 3 for proximal decoding, the following simulation results are based on Monte Carlo simulations and the BER and FER curves have been generated by producing at least 100 frame errors for each data point, except in cases where this is explicitly specified otherwise.

4.3.1 Choice of Parameters

The first two parameters to be investigated are the penalty parameter μ and the over-relaxation parameter ρ. A first approach to get some indication of the values that might be chosen for these parameters is to look at how the decoding performance depends on them. The FER is plotted as a function of μ and ρ in figure 4.2, for three different SNRs. The code chosen for this examination is a (3,6) regular LDPC code with $n = 204$ and $k = 102$ [Mac23, 204.33.484]. When varying μ, ρ is set to 1 and when varying ρ, μ is set to 5. The maximum number of iterations K is set to 200 and ϵ_{dual} and ϵ_{pri} to 10^{-5}. The behavior that can be observed is very similar to that of the parameter γ in proximal decoding, analyzed in section 3.2. A single optimal value giving optimal performance does not exist; rather, as long as the value is chosen within a certain range, the performance is approximately equally good.

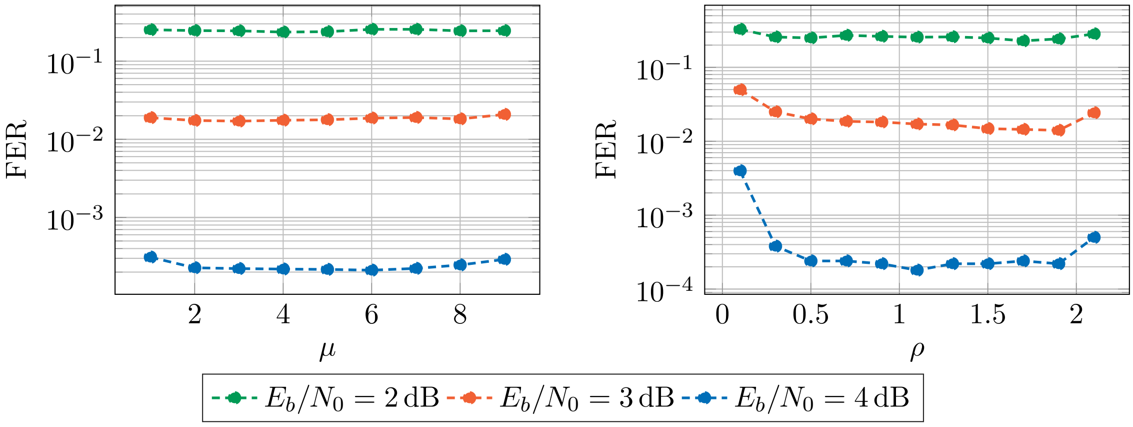


Figure 4.2: Dependence of the decoding performance on the parameters μ and ρ. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

To aid in the choice of the parameters, an additional criterion can be used: the number of iterations performed for a decoding operation. This is directly related to the time needed to decode a received vector $\boldsymbol{y}$, which the aim is to minimize. Figure 4.3 shows the average number of iterations over 1000 decodings, as a function of ρ. This time the SNR is kept constant at 4 dB and the parameter μ is varied. The values chosen for the rest of the parameters are the same as before. It is visible that choosing a large value for ρ as well as a small value for μ minimizes the average number of iterations and thus the average run time of the decoding process. The same behavior can be observed when looking at various different codes, as shown in figure 4.8.

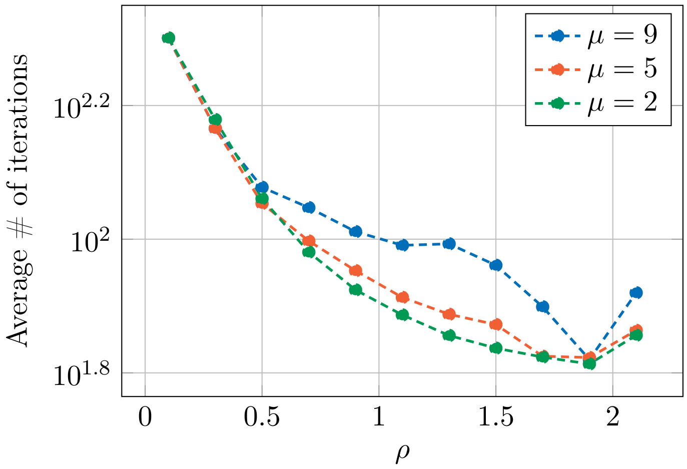


Figure 4.3: Dependence of the average number of iterations required on μ and ρ for $E_b/N_0 =$ 4 dB. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

To get an estimate for the maximum number of iterations K necessary, the average error during decoding can be used. This is shown in figure 4.4 as an average of 100 000 decodings.

μ is set to 5 and ρ is set to 1 and the rest of the parameters are again chosen as $\epsilon_{\text{pri}} = 10^{-5}$ and $\epsilon_{\text{dual}} = 10^{-5}$. Similarly to the results in section 3.2.1, a dip is visible around the 20 iteration mark. This is due to the fact that as the number of iterations increases, more and more decodings converge, leaving only the mistaken ones to be averaged. The point at which the wrong decodings start to become dominant and the decoding performance does not increase any longer is largely independent of the SNR, allowing the maximum number of iterations to be chosen without considering the SNR.

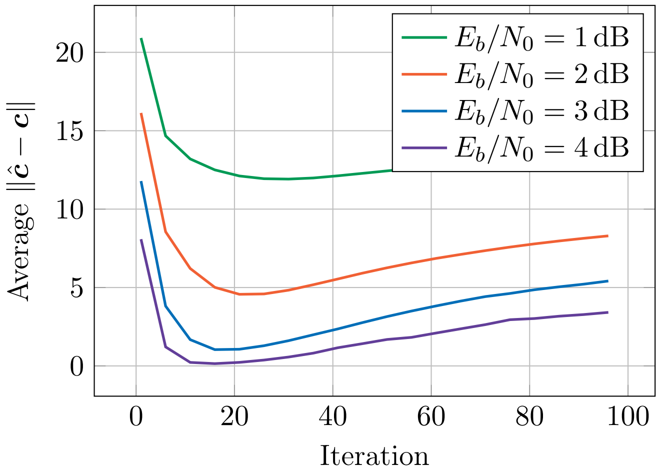


Figure 4.4: Average error for 100 000 decodings. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

The last two parameters remaining to be examined are the tolerances for the stopping criterion of the algorithm, ϵ_{pri} and ϵ_{dual}. These are both set to the same value ϵ. The effect of their value on the decoding performance is visualized in figure 4.5. All parameters except ϵ_{pri} and ϵ_{dual} are kept constant, with $\mu = 5$, $\rho = 1$ and $E_b/N_0 = 4\,\text{dB}$ and performing a maximum of 200 iterations. A lower value for the tolerance initially leads to a dramatic decrease in the FER, this effect fading as the tolerance becomes increasingly lower.

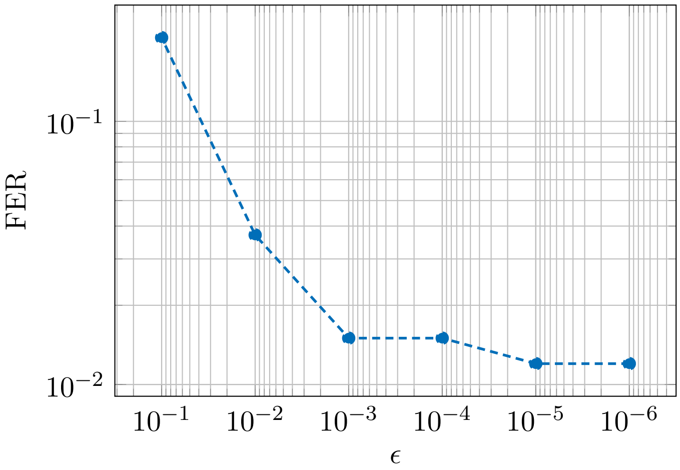


Figure 4.5: Effect of the value of the parameters ϵ_{pri} and ϵ_{dual} on the FER. (3,6) regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

In conclusion, the parameters μ and ρ should be chosen comparatively small and large, respectively, to reduce the average runtime of the decoding process, while keeping them within a certain range as to not compromise the decoding performance. The maximum number of iterations performed can be chosen independently of the SNR. Finally, small values should be given to the parameters ϵ_{pri} and ϵ_{dual} to achieve the lowest possible error rate.

4.3.2 Decoding Performance

In figure 4.6, the simulation results for the "Margulis" LDPC code ($n = 2640$, $k = 1320$) presented by Barman et al. in [Bar+13] are compared to the results from the simulations conducted in the context of this thesis. The parameters chosen were $\mu = 3.3$, $\rho = 1.9$, $K = 1000$, $\epsilon_{\text{pri}} = 10^{-5}$ and $\epsilon_{\text{dual}} = 10^{-5}$, the same as in [Bar+13]. The two FER curves are practically identical. Also shown is the curve resulting from BP decoding, performing 1000 iterations. The two algorithms perform relatively similarly, staying within 0.5 dB of one another.

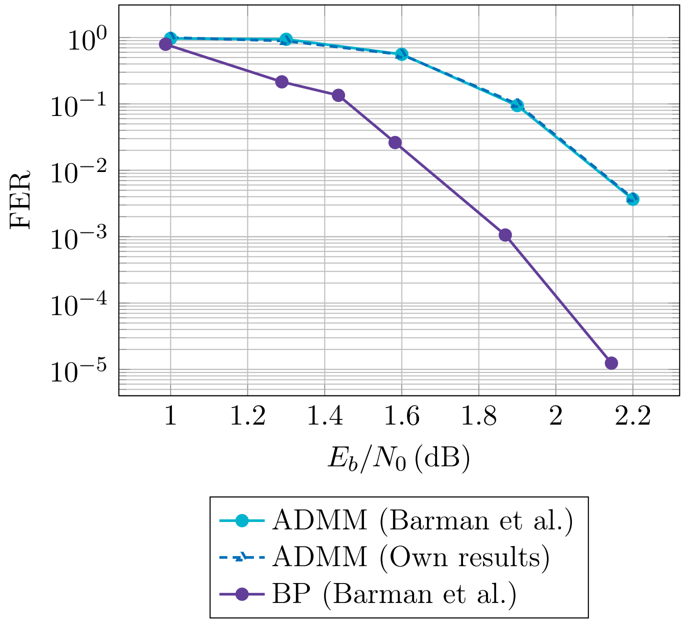


Figure 4.6: Comparison of datapoints from Barman et al. with own simulation results. "Margulis" LDPC code with $n = 2640$, $k = 1320$ [Mac23, Margulis2640.1320.3]

In figure 4.9, FER curves for LP decoding using ADMM and BP are shown for various codes. To ensure comparability, in all cases the number of iterations was set to $K = 200$. The values of the other parameters were chosen as $\mu = 5$, $\rho = 1$, $\epsilon_{\text{pri}} = 10^{-5}$ and $\epsilon_{\text{dual}} = 10^{-5}$. Comparing the simulation results for the different codes, it is apparent that the difference in decoding performance depends on the code being considered. For all codes considered here, however, the performance of LP decoding using ADMM comes close to that of BP, again staying withing approximately 0.5 dB.

4.3.3 Computational Performance

In terms of time complexity, the three steps of the decoding algorithm in equations (4.6) - (4.8) have to be considered. The $\tilde{\boldsymbol{c}}$- and $\boldsymbol{u}_j$-update steps are $\mathcal{O}(n)$ [Bar+13, Sec. III. C.]. The complexity of the $\boldsymbol{z}_j$-update step depends on the projection algorithm employed. Since for the implementation completed for this work the projection algorithm presented in [Bar+13] is used, the $\boldsymbol{z}_j$-update step also has linear time complexity.

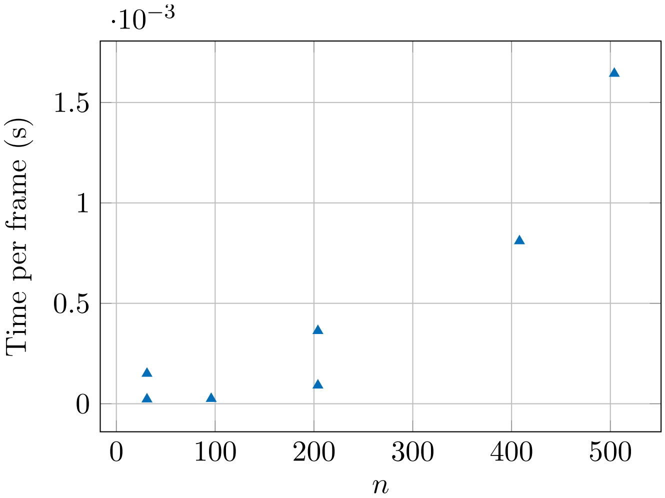
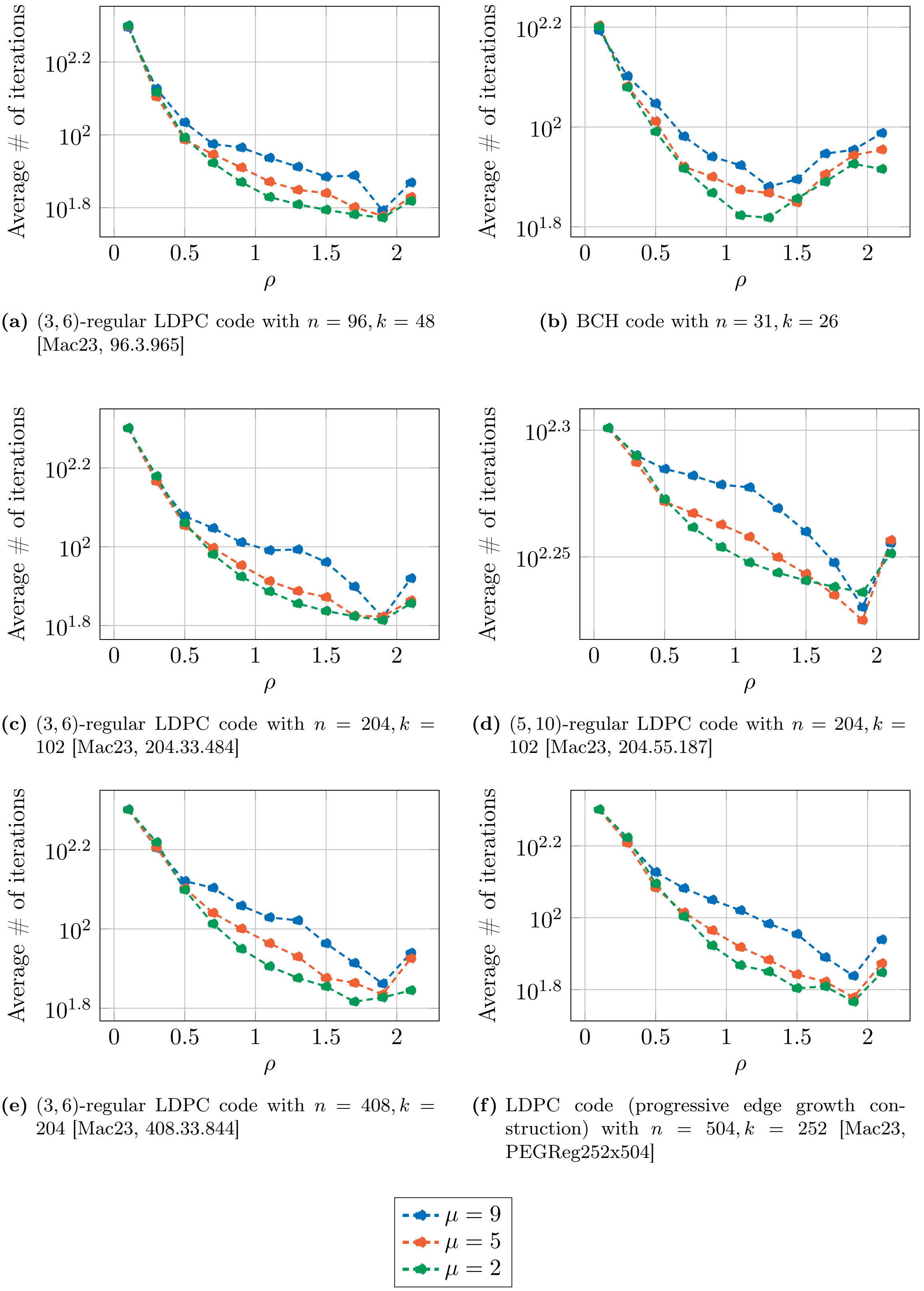


Figure 4.7: Timing requirements of the LP decoding using ADMM implementation

Simulation results from a range of different codes can be used to verify this analysis. Figure 4.7 shows the average time needed to decode one frame as a function of its length. The codes used for this consideration are the same as in section 3.2.4 The results are necessarily skewed because these vary not only in their length, but also in their construction scheme and rate. Additionally, different optimization opportunities arise depending on the length of a code, since for smaller codes dynamic memory allocation can be completely omitted. This may explain why the datapoint at $n = 504$ is higher then would be expected with linear behavior. Nonetheless, the simulation results roughly match the expected behavior following from the theoretical considerations.



(a) $(3, 6)$-regular LDPC code with $n = 96, k = 48$ [Mac23, 96.3.965]

(b) BCH code with $n = 31, k = 26$

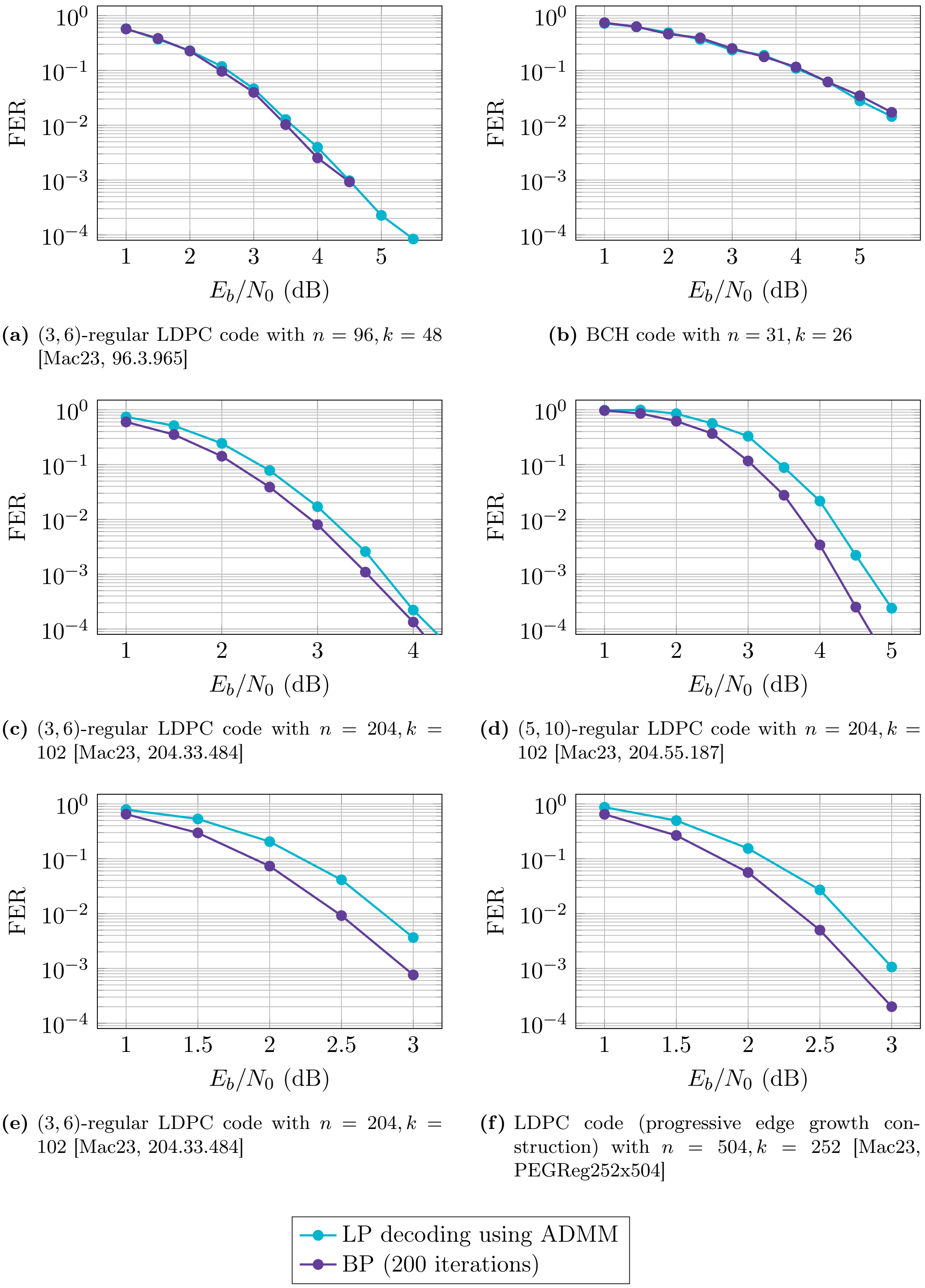
(c) $(3, 6)$-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(d) $(5, 10)$-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.55.187]

(e) $(3, 6)$-regular LDPC code with $n = 408, k = 204$ [Mac23, 408.33.844]

(f) LDPC code (progressive edge growth construction) with $n = 504, k = 252$ [Mac23, PEGReg252x504]

Figure 4.8: Dependence of the average number of iterations required on the parameters μ and ρ for $E_b/N_0 = 4\,\text{dB}$ for various codes



(a) (3, 6)-regular LDPC code with $n = 96, k = 48$ [Mac23, 96.3.965]

(b) BCH code with $n = 31, k = 26$

(c) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(d) (5, 10)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.55.187]

(e) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(f) LDPC code (progressive edge growth construction) with $n = 504, k = 252$ [Mac23, PEGReg252x504]

Figure 4.9: Comparison of the decoding performance of LP decoding using ADMM and BP for various codes

5 Comparison of Proximal Decoding and LP Decoding using ADMM

In this chapter, proximal decoding and LP Decoding using ADMM are compared. First, the two algorithms are studied on a theoretical basis. Subsequently, their respective simulation results are examined, and their differences are interpreted based on their theoretical structure.

5.1 Theoretical Comparison

ADMM and the proximal gradient method can both be expressed in terms of proximal operators [PB14, Sec. 4.4]. Additionally, the two algorithms show some striking similarities with regard to their general structure and the way in which the minimization of the respective objective functions is accomplished.

The LP decoding problem in equation (4.4) can be slightly rewritten using the *indicator functions* $g_j : \mathbb{R}^{d_j} \rightarrow \{0, +\infty\}$, $j \in \mathcal{J}$ for the polytopes $\mathcal{P}_{d_j}$, $j \in \mathcal{J}$, defined as

$$g_j\left(\boldsymbol{t}\right) := \begin{cases} 0, & \boldsymbol{t} \in \mathcal{P}_{d_j} \\ +\infty, & \boldsymbol{t} \notin \mathcal{P}_{d_j} \end{cases},$$

by moving the constraints into the objective function, as shown in figure 5.1b. The objective functions of the two problems are similar in that they both comprise two parts: one associated to the likelihood that a given codeword was sent, arising from the channel model, and one associated to the constraints the decoding process is subjected to, arising from the code used. Both algorithms are composed of an iterative approach consisting of two alternating steps, each minimizing one part of the objective function.

Their major difference is that while with proximal decoding the constraints are regarded in a global context, considering all parity checks at the same time, with ADMM each parity check is considered separately and in a more local context (line 4 in both algorithms). Furthermore, while with proximal decoding the step considering the constraints is realized using gradient descent - amounting to an approximation - with ADMM it reduces to a number of projections onto the parity polytopes $\mathcal{P}_{d_j}$, $j \in \mathcal{J}$, which always provide exact results.

The contrasting treatment of the constraints (global and approximate with proximal decoding as opposed to local and exact with LP decoding using ADMM) also leads to different prospects when the decoding process gets stuck in a local minimum. With proximal decoding this occurs due to the approximate nature of the calculation, whereas with LP decoding it occurs due to the approximate formulation of the constraints - independent of the optimization method itself. The advantage which arises because of this when employing LP decoding is the ML certificate property: when a valid codeword is returned, it is also the ML codeword. This means that additional redundant parity-checks can be added successively until the codeword returned is valid and thus the ML solution is found [TS06, Sec. IV.].

$$\begin{aligned} \text{minimize}\quad & \underbrace{L\left(\boldsymbol{y} \mid \tilde{\boldsymbol{x}}\right)}_{\text{Likelihood}} + \underbrace{\gamma h\left(\tilde{\boldsymbol{x}}\right)}_{\text{Constraints}} \\ \text{subject to}\quad & \tilde{\boldsymbol{x}} \in \mathbb{R}^n \end{aligned}$$

```
Initialize $\boldsymbol{r}, \boldsymbol{s}, \omega, \gamma$
while stopping critierion unfulfilled do
    $\boldsymbol{r} \leftarrow \boldsymbol{r} + \omega \nabla L\left(\boldsymbol{y} \mid \boldsymbol{s}\right)$
    $\boldsymbol{s} \leftarrow \textbf{prox}_{\gamma h}\left(\boldsymbol{r}\right)$

end while
return $\boldsymbol{s}$
```

(a) Proximal decoding

$$\begin{aligned} \text{minimize}\quad & \underbrace{\boldsymbol{\gamma}^{\mathrm{T}} \tilde{\boldsymbol{c}}}_{\text{Likelihood}} + \underbrace{\sum\nolimits_{j \in \mathcal{J}} g_j\left(\boldsymbol{T}_j \tilde{\boldsymbol{c}}\right)}_{\text{Constraints}} \\ \text{subject to}\quad & \tilde{\boldsymbol{c}} \in [0,1]^n \end{aligned}$$

```
Initialize $\tilde{\boldsymbol{c}}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{\gamma}, \rho$
while stopping criterion unfulfilled do
    $\tilde{\boldsymbol{c}} \leftarrow \text{argmin}_{\tilde{\boldsymbol{c}}} \left(\boldsymbol{\gamma}^{\mathrm{T}} \tilde{\boldsymbol{c}} + \frac{\rho}{2} \sum_{j \in \mathcal{J}} \left\| \boldsymbol{T}_j \tilde{\boldsymbol{c}} - \boldsymbol{z}_j + \boldsymbol{u}_j \right\|\right)$
    $\boldsymbol{z}_j \leftarrow \textbf{prox}_{g_j}\left(\boldsymbol{T}_j \tilde{\boldsymbol{c}} + \boldsymbol{u}_j\right), \quad \forall j \in \mathcal{J}$
    $\boldsymbol{u}_j \leftarrow \boldsymbol{u}_j + \tilde{\boldsymbol{c}} - \boldsymbol{z}_j, \quad \forall j \in \mathcal{J}$
end while
return $\tilde{\boldsymbol{c}}$
```

(b) LP decoding using ADMM

Figure 5.1: Comparison of proximal decoding and LP decoding using ADMM

In terms of time complexity, the two decoding algorithms are comparable. Each of the operations required for proximal decoding can be performed in $\mathcal{O}\left(n\right)$ time for LDPC codes (see section 3.2.4). The same is true for LP decoding using ADMM (see section 4.3.3). Additionally, both algorithms can be understood as message-passing algorithms, LP decoding using ADMM as similarly to [Bar+13, Sec. III. D.] and [ZS13, Sec. II. B.], and proximal decoding by starting with algorithm 3.1, substituting for the gradient of the code-constraint polynomial and separating the $\boldsymbol{s}$ update into two parts. The algorithms in their message-passing form are depicted in figure 5.2. $M_{j \to i}$ denotes a message transmitted from CN j to VN i. This message passing structure means that both algorithms can be implemented very efficiently, as the update steps can be performed in parallel for all CNs and for all VNs, respectively.

In conclusion, the two algorithms have a very similar structure, where the parts of the objective function relating to the likelihood and to the constraints are minimized in an alternating fashion. With proximal decoding this minimization is performed for all constraints at once in an approximative manner, while with LP decoding using ADMM it is performed for each constraint individually and with exact results. In terms of time complexity, both algorithms are linear with respect to n and are heavily parallelizable.

5.2 Comparison of Simulation Results

The decoding performance of the two algorithms is compared in figure 5.4 in form of the FER. Shown as well is the performance of the improved proximal decoding algorithm presented in section 3.3. The FER resulting from decoding using BP and, wherever available,

```
Initialize $\boldsymbol{r}, \boldsymbol{s}, \omega, \gamma$
while stopping critierion unfulfilled do
    for j in $\mathcal{J}$ do
        $p_j \leftarrow \prod_{i \in N_c(j)} r_i$
        $M_{j \to i} \leftarrow p_j^2 - p_j$
    end for
    for i in $\mathcal{I}$ do
        $s_i \leftarrow \Pi_\eta \left( s_i + \gamma \left( 4 \left( s_i^2 - 1 \right) s_i + \frac{4}{s_i} \sum_{j \in N_v(i)} M_{j \to i} \right) \right)$
        $r_i \leftarrow r_i + \omega \left( s_i - y_i \right)$
    end for
end while
return $\boldsymbol{s}$
```

(a) Proximal decoding

```
Initialize $\tilde{\boldsymbol{c}}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{\gamma}, \rho$
while stopping critierion unfulfilled do
    for j in $\mathcal{J}$ do
        $\boldsymbol{z}_j \leftarrow \Pi_{P_{d_j}} \left( \boldsymbol{T}_j \tilde{\boldsymbol{c}} + \boldsymbol{u}_j \right)$
        $\boldsymbol{u}_j \leftarrow \boldsymbol{u}_j + \boldsymbol{T}_j \tilde{\boldsymbol{c}} - \boldsymbol{z}_j$
        $M_{j \to i} \leftarrow \left( z_j \right)_i - \left( u_j \right)_i, \quad \forall i \in N_c(j)$
    end for
    for i in $\mathcal{I}$ do
        $\tilde{c}_i \leftarrow \frac{1}{d_i} \left( \sum_{j \in N_v(i)} M_{j \to i} - \frac{\gamma_i}{\mu} \right)$
    end for
end while
return $\tilde{\boldsymbol{c}}$
```

(b) LP decoding using ADMM

Figure 5.2: Proximal decoding and LP decoding using ADMM as message passing algorithms

the FER of ML decoding, taken from [Hel+23], are plotted as a reference. The parameters chosen for the proximal and improved proximal decoders are $\gamma = 0.05$, $\omega = 0.05$, $K = 200$, $\eta = 1.5$ and $N = 12$. The parameters chosen for LP decoding using ADMM are $\mu = 5$, $\rho = 1$, $K = 200$, $\epsilon_{\text{pri}} = 10^{-5}$ and $\epsilon_{\text{dual}} = 10^{-5}$. For all codes considered within the scope of this work, LP decoding using ADMM consistently outperforms both proximal decoding and the improved version, reaching very similar performance to BP. The decoding gain heavily depends on the code, evidently becoming greater for codes with larger n and reaching values of up to 2 dB.

These simulation results can be interpreted with regard to the theoretical structure of the decoding methods, as analyzed in section 5.1. The worse performance of proximal decoding is somewhat surprising, considering the global treatment of the constraints in contrast to the local treatment in the case of LP decoding using ADMM. It may be explained, however, in the context of the nature of the calculations performed in each case. With proximal decoding, the calculations are approximate, leading to the constraints never being quite satisfied. With LP decoding using ADMM, the constraints are fulfilled for each parity check individually after each iteration of the decoding process. A further contributing factor might be the structure of the optimization process, as the alternating minimization with respect to the same variable leads to oscillatory behavior, as explained in section 3.2.3. It should be noted that while in this thesis proximal decoding was examined with respect to its performance in AWGN channels, in [WT22] it is presented as a method applicable to non-trivial channel models such as LDPC-coded massive MIMO channels, perhaps broadening its usefulness beyond what is shown here.

The timing requirements of the decoding algorithms are visualized in figure 5.3. The datapoints have been generated by evaluating the metadata from FER and BER simulations and using the parameters mentioned earlier when discussing the decoding performance. The codes considered are the same as in sections 3.2.4 and 4.3.3. While the ADMM implementation seems to be faster than the proximal decoding and improved proximal decoding implementations, inferring some general behavior is difficult in this case. This is because of the comparison of actual implementations, making the results dependent on factors such as the grade of optimization of each of the implementations. Nevertheless, the run time of both the proximal decoding and the LP decoding using ADMM implementations is similar, and both are reasonably performant, owing to the parallelizable structure of the algorithms.

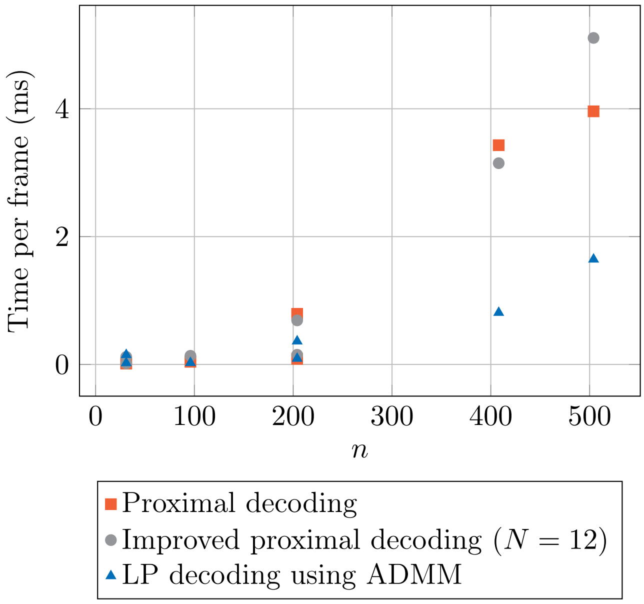
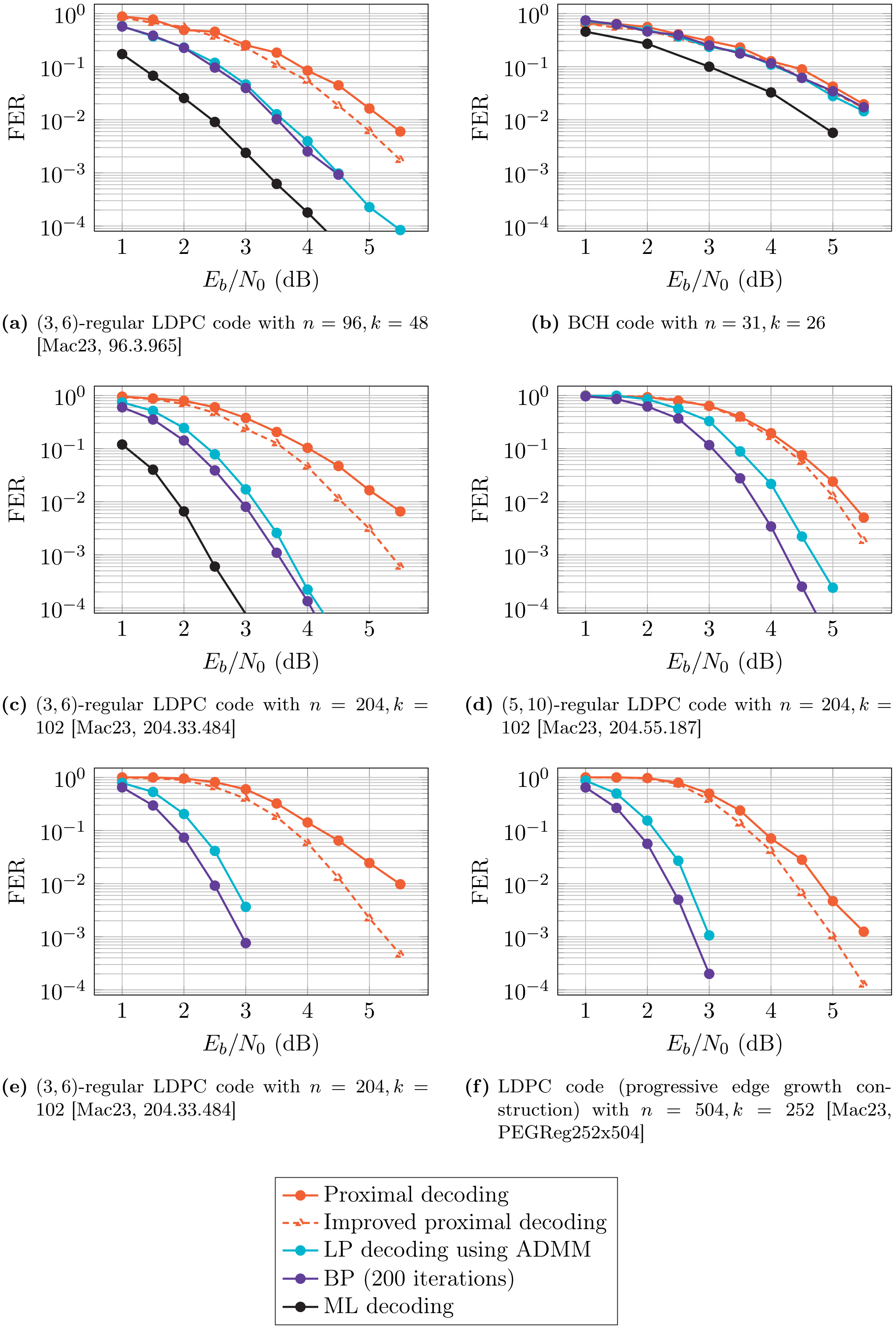


Figure 5.3: Comparison of the timing requirements of the different decoder implementations



(a) (3, 6)-regular LDPC code with $n = 96, k = 48$ [Mac23, 96.3.965]

(b) BCH code with $n = 31, k = 26$

(c) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(d) (5, 10)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.55.187]

(e) (3, 6)-regular LDPC code with $n = 204, k = 102$ [Mac23, 204.33.484]

(f) LDPC code (progressive edge growth construction) with $n = 504, k = 252$ [Mac23, PEGReg252x504]

Figure 5.4: Comparison of the decoding performance of the different decoder implementations for various codes

6 Conclusion and Outlook

In the context of this thesis, two decoding algorithms were considered: proximal decoding and LP decoding using ADMM. The two algorithms were first analyzed individually, before comparing them based on simulation results as well as on their theoretical structure.

For proximal decoding, the effect of each parameter on the behavior of the decoder was examined, leading to an approach to optimally choose the value of each parameter. The convergence properties of the algorithm were investigated in the context of the relatively high decoding failure rate, to derive an approach to correct possibly wrong components of the estimate. Based on this approach, an improvement of proximal decoding was suggested, leading to a decoding gain of up to 1 dB, depending on the code and the parameters considered.

For LP decoding using ADMM, the circumstances brought about by the LP relaxation were first explored. The decomposable nature arising from the relocation of the constraints into the objective function itself was recognized as the major driver in enabling an efficient implementation of the decoding algorithm. Based on simulation results, general guidelines for choosing each parameter were derived. The decoding performance, in form of the FER, of the algorithm was analyzed, observing that LP decoding using ADMM nearly reaches that of BP, staying within approximately 0.5 dB depending on the code in question.

Finally, strong parallels were discovered with regard to the theoretical structure of the two algorithms, both in the constitution of their respective objective functions as well as in the iterative approaches used to minimize them. One difference noted was the approximate nature of the minimization in the case of proximal decoding, leading to the constraints never being truly satisfied. In conjunction with the alternating minimization with respect to the same variable, leading to oscillatory behavior, this was identified as a possible cause of its comparatively worse decoding performance. Furthermore, both algorithms were expressed as message passing algorithms, illustrating their similar computational performance.

While the modified proximal decoding algorithm presented in section 3.3 shows some promising results, further investigation is required to determine how different choices of parameters affect the decoding performance. Additionally, a more mathematically rigorous foundation for determining the potentially wrong components of the estimate is desirable. A different method to improve proximal decoding might be to use moment-based optimization techniques such as *Adam* [KB14] to try to mitigate the effect of local minima introduced in the objective function as well as the adversarial structure of the minimization when employing proximal decoding.

Another area benefiting from future work is the expansion of the ADMM based LP decoder into a decoder approximating ML performance, using *adaptive LP decoding*. With this method, the successive addition of redundant parity checks is used to mitigate the decoder becoming stuck in erroneous solutions introduced due the relaxation of the constraints of the LP decoding problem [TS06].

List of Abbreviations

ADMM	Alternating direction method of multipliers
AWGN	Additive white Gaussian noise
BER	Bit error rate
BP	Belief propagation
BPSK	Binary phase-shift keying
CN	Check node
FER	Frame error rate
LDPC	Low-density parity-check
LLR	Log-likelihood ratio
LP	Linear programming
MAP	Maximum a posteriori
MIMO	Multiple-input multiple-output
ML	Maximum likelihood
PDF	Probability density function
PMF	Probability mass function
SNR	Signal-to-noise ratio
VN	Variable node

Bibliography

[Bar+13] Siddharth Barman et al. “Decomposition Methods for Large Scale LP Decoding”. In: *IEEE Transactions on Information Theory* 59.12 (2013), pp. 7870–7886.

[Boy+11] Stephen Boyd et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends in Machine learning* 3.1 (2011), pp. 1–122.

[BT97] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization.* Vol. 6. Athena scientific Belmont, MA, 1997.

[Fel03] J. Feldman. “Decoding error-correcting codes via linear programming”. PhD thesis. MIT, 2003.

[FWK05] J. Feldman, M.J. Wainwright, and D.R. Karger. “Using linear programming to Decode Binary linear codes”. In: *IEEE Transactions on Information Theory* 51.3 (2005), pp. 954–972.

[Gen+20] Florian Gensheimer et al. “A Reduced-Complexity Projection Algorithm for ADMM-Based LP Decoding”. In: *IEEE Transactions on Information Theory* 66.8 (2020), pp. 4819–4833.

[Hel+23] Michael Helmling et al. *Database of Channel Codes and ML Simulation Results*. Apr. 2023. URL: `https://www.uni-kl.de/channel-codes`.

[KB14] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[Mac23] David J.C. MacKay. *Encyclopedia of Sparse Graph Codes*. Apr. 2023. URL: `http://www.inference.org.uk/mackay/codes/data.html`.

[Mac99] D.J.C. MacKay. “Good error-correcting codes based on very sparse matrices”. In: *IEEE Transactions on Information Theory* 45.2 (1999), pp. 399–431.

[PB14] Neal Parikh and Stephen Boyd. “Proximal Algorithms”. In: *Found. Trends Optim.* 1.3 (Jan. 2014), pp. 127–239.

[RL09] William Ryan and Shu Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[TS06] Mohammad H. Taghavi and Paul H. Siegel. “Adaptive Linear Programming Decoding”. In: *2006 IEEE International Symposium on Information Theory.* 2006, pp. 1374–1378.

[Von08] Pascal O. Vontobel. “Interior-point algorithms for linear-programming decoding”. In: *2008 Information Theory and Applications Workshop*. 2008, pp. 433–437.

[WT22] Tadashi Wadayama and Satoshi Takabe. “Proximal Decoding for LDPC Codes”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* advpub (2022), 2022TAP0002.

[Zha+19] Ming-Min Zhao et al. “Decoding Binary Linear Codes Using Penalty Dual Decomposition Method”. In: *IEEE Communications Letters* 23.6 (2019), pp. 958–962.

[ZS13] Xiaojie Zhang and Paul H. Siegel. “Efficient iterative LP decoding of LDPC codes with alternating direction method of multipliers”. In: *2013 IEEE International Symposium on Information Theory*. 2013, pp. 1501–1505.