JACOBS UNIVERSITY

LDPC Codes Incorporating Source, Noise, and Channel Memory

by

Nazia Sarwat Islam

a thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
in Electrical Engineering

Approved Dissertation Committee

Prof. Dr.-Ing. Werner Henkel
Jacobs University Bremen

Dr. rer. nat. habil. Mathias Bode
Jacobs University Bremen

Prof. Dr.-Ing. habil. Volker Kühn
Universität Rostock

Prof. Dr. Francis C. M. Lau
The Hong Kong Polytechnic University

Date of Defense: March 8, 2022

Computer Science and Electrical Engineering

Abstract

This thesis discusses how memory of the source, of disturbances, or of the channel can be efficiently dealt with inside the decoding of LDPC codes. Furthermore, how such codes can be optimized for including source memory is also presented.

At the source, the memory is modeled via a Markov chain. The transition probabilities of the model are used at the decoder to estimate the source symbols. Although computed at the decoder, this information is considered to be a-priori information. The a-priori LLR can be directly incorporated into the Tanner graph via directed edges between the variable nodes, a novel simplified computation which provides equal performance to existing methods is shown. A Turbo-like scheme is also proposed where a BCJR and an LDPC decoder decode the source and received sequences iteratively, each utilizing extrinsic information computed by the other decoder. The Turbo decoding scheme performs the best at low SNRs. Subsequently, a code design algorithm is provided for obtaining optimized codes for the decoding model with direct additional links between variable nodes. For the optimization, density evolution is used. The optimized codes provide steeper performance curves than non-optimized ones.

Thereafter, impulse noise with memory is investigated, which is modeled by the Middleton Class-A model. A Markov model provides the transition probabilities between background and impulsive noise states. A Viterbi decoder estimates the noise sequence and an LDPC decoder estimates the transmitted symbols. Information is iterated between the decoders to improve the overall correction at the receiver. Possibilities for computing the noise states directly at the decoder are also investigated. However, the noise-memory cannot be directly incorporated into the Tanner graph. Lastly, a method is proposed to mitigate channel memory which causes inter-symbol interference using an LDPC decoder. A decision-feedback equalization like structure is used in which the intermediate LDPC decoder results are used for equalization.

Acknowledgements

This doctoral thesis has been possible due to the support of many people. I would like to acknowledge their contributions.

Firstly, I would like to express my sincerest gratitude and appreciation to my supervisor Prof. Dr.-Ing. Werner Henkel. Throughout the years, he has always offered invaluable support on every front and has been a great mentor. I would also like to extend my gratitude to Dr. rer. nat. habil. Mathias Bode for conversations and support provided, both academic and motivational. I would like to thank Prof. Dr.-Ing. habil. Volker Kühn and Prof. Dr. Francis C. M. Lau for being a part of the thesis committee. Special thanks go to my colleagues Dawit Nigatu, Oana Graur, Abdul Wakeel, Khodr Saifan, Vladimir Cherkashyn, Hashem Ziyabar and Ahmed Leghari for all the conversations and support during my time at Jacobs. Additionally, I would like to thank Ahmed Leghari and Guanan Mu for their work on topics related to this thesis.

I would like to thank my friends, for being a second family to me and for always being there, whenever I needed. To my friends: Hannah, Denise, Prashant, Rakhshan, Noni, Ahsan, Nabeera, Shucheta - I owe a debt of gratitude for making my life one that I deeply cherish. To Mauricio, thank you for always being there and listening, you were a great strength to me. To Sören, thank you for your patience, company, and constant encouragement, 2020-2021 wasn't as hard because of you.

To my family, thank you, for everything. To Lamia, thank you for always having my back. To mom and dad, thank you for always supporting me, you are my greatest strength. It goes without saying, but I would not be here without you. This is dedicated to you.

Contents

Chapter 1

Introduction

1.1 Motivation and Objectives

The motivation behind this thesis is understanding and extending iterative LDPC decoding to exploit memory or dependency present in different parts of the transmission chain. LDPC codes are capacity approaching codes. In the 1990s, simulations of iterative coding techniques provided possibilities for channel capacity approaching performance for both Turbo and LDPC codes, providing substantial gains over existing methods of channel coding. At present, LDPC codes are being implemented in current standards, e.g., for deep space communications, 5G telecommunication standards etc.

Here, source, noise, and channel memory are incorporated into LDPC codes. Realization of memory in each of these components of the transmission chain is different and requires separate handling. We have investigated an LDPC decoder for decoding sources with memory, for mitigating impulse noise with memory, and for canceling inter-symbol interference; a manifestation of dependency arising from the channel transfer function. A code optimization procedure is also presented which takes into account the modified decoder for handling source memory.

1.2 Thesis Contributions

The contributions of this thesis are listed below.

- Different methods for using source dependencies in LDPC decoding are provided and compared.
- Optimized codes for sources which have dependency modeled by a discrete time Markov chain are found by utilizing density evolution. The convergence constraint for the density

evolution procedure is reformulated to incorporate source memory.

- An LDPC decoding scheme for mitigating Middleton Class-A impulse noise model with memory is provided.
- An LDPC based decision-feedback-equalizer for ISI cancellation is shown.

Together, these four parts form our analysis of LDPC decoder interaction with the other parts of the transmission system.

1.3 Structure of the Thesis

In the following, we provide short descriptions of the contents of the chapters of this thesis.

Chapter 2: Basic Concepts

In this chapter, we provide the framework for all concepts, methods, and algorithms used in this thesis. We provide an overview of Markov models, channel codes with a focus on LDPC codes as well as Turbo codes.

Chapter 3: LDPC Decoding for Sources with Memory

This chapter focuses on the modification of the decoder for a data source modeled by a Markov chain. A source sequence generated by such a model implies that information from neighboring variable nodes can be used for decoding, for a systematic code. Following the discrete time Markov chain which models the source, the information between neighboring variable nodes are directed in a 'left-to-right' orientation. We provide three decoding techniques for utilizing this information and provide simulation results for each showing the gains obtained by using the dependency. We investigate order-1 and order-2 Markov chains for the simulations.

Chapter 4: Optimization of LDPC Codes for Sources with Memory

In this chapter, density evolution based techniques are used for finding optimized codes for sources modeled by a Markov chain. The 'left-to-right' edges between information variable-nodes changes the typical mutual information calculation equations, replacing them with a quadratic expression in the variable-node degree distribution polynomial. The BER curves of the optimized codes are steeper compared to codes optimized for the AWGN channel, even

when the standard codes have lower SNR thresholds at which convergence begins.

Chapter 5: LDPC Codes in Impulse Noise with Memory

In this chapter, we consider a channel disturbed by impulse noise with memory, modeled by Middleton class-A model. The memory can be expressed as a Markov model and we show 4 different methods of mitigation at the decoder. A Turbo-like scheme, using a Viterbi decoder for estimating the noise state sequence is proposed. The memory in the noise process is successfully used for mitigation of impulse noise.

Chapter 6: LDPC-based Decision Feedback Equalization

In this chapter, an LDPC decoder is incorporated into decision feedback equalization. We present an iterative method in which intermediate decoding results from an LDPC decoder are used for cancelling the ISI, effectively this is an iterative DFE. The LDPC decoder also consequently benefits from better channel estimates leading to modified intrinsic information, the interplay between the two decoders is in a Turbo-like fashion.

Chapter 7: Conclusion

In this chapter, we summarize the methods implemented for each memory aspect investigated. We discuss the results obtained and the applicability of using LDPC decoding in conjunction with processing in other parts of the communication chain. We conclude the thesis by outlining the future aspects of investigation.

Chapter 2

Basic Concepts

In this chapter, we provide a technical overview of the methods used in the rest of the work. The overall thesis topic is an investigation of dependencies in different parts of the transmission block, as illustrated in Fig. 2.1. For both the source and noise realization of dependency, Markov models are used. We begin our description there. We then move on to providing an overview of LDPC codes, which are used as the standard channel code in this thesis.

In Fig. 2.1, a block diagram of a realization of a digital transmission scheme is shown.

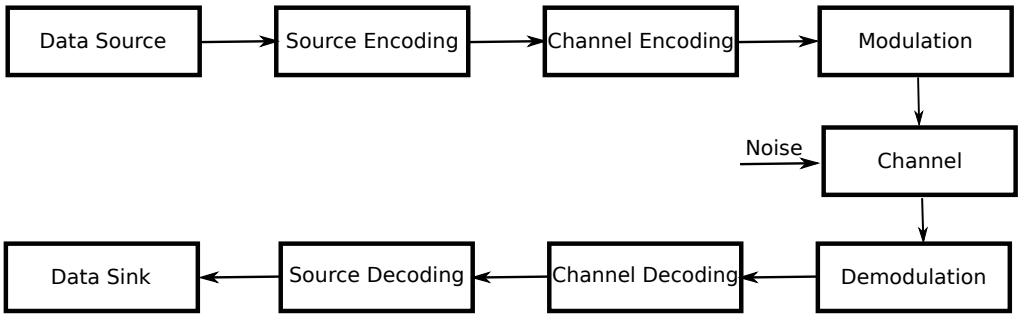


Figure 2.1: Block-diagram of a transmission system

We have divided the processes in to three sub-divisions, on the transmission side, which address the following:

- Data handling before transmission,
- Noise environment,
- Channel impulse response causing inter-symbol interference.

The topic of this thesis is treating *memory* or *dependencies* present in these three parts at the channel decoder on the receiver side. As a channel code, we have used low-density parity-check (LDPC) codes consistently, as well as always using BPSK modulation. We now briefly

present the handling of dependencies in each of the three sub-divisions.

Data Handling before Transmission

In this part, the source data was modeled as resulting from a Markov model. This results in the source information bits having a memory between them. Without doing separate source encoding, we directly performed channel encoding, i.e., computed the parities for the source-optimized channel code. On the receiver side, we used the source model directly in the channel decoder. Simulations for three decoder realizations are shown and results are compared. For designing source optimized LDPC codes based on the modified belief propagation decoding, computational changes relating to LLR density transformations reflecting structural changes in the Tanner graph have to be formulated. Subsequently, optimized codes are obtained using all relevant constraints of the source optimized channel code and results are compared with codes obtained by standard density evolution methods.

Noise Environment

This portion of the thesis is focused on impulse noise mitigation. We consider an impulse noise model with memory, again described by a Markov model. We focus on deducing the active noise environment (impulsive or background Gaussian) during transmission at the receiver, in order to mitigate the extreme effects of impulse noise on system performance. We implemented a Turbo-like scheme using a Viterbi decoder and an LDPC decoder iterating between them to deduce the noise states and using the variance of the estimated noise state for decoding the transmitted symbols, respectively.

Channel Impulse Response causing Inter-symbol Interference

For this section of the thesis, we investigated ISI cancellation in the Tanner graph by using a decision-feedback equalizer structure within the LDPC decoder. This method provides an iterative approach as well, however, in contrast to Turbo-equalization, this method has lower complexity.

We now move on to describing the required models, methods, and algorithms for implementing the above mentioned memory-incorporating LDPC designs.

2.1 Markov Models

Markov models are stochastic models describing systems of random variables. They were first proposed by A. A. Markov in 1913, an English translation of the original (Russian) paper is available in [1]. Models are represented using a state-space description, i.e., random variables representing which state the model is in at a specified time instant. A model is given by the probabilities which govern the transitions between states and probabilities of observing specific outputs depending on the states. We provide some basic definitions in the following.

Markov Property

A stochastic or probabilistic process has the Markov property if the next state of the system only depends on the current state. In other words, the Markov property implies conditional independence of the random variable at time $t+1$ on all past states, it is only dependent on the present state, at time t.

Let S_t denote the random variable describing the states of a system at time t and let the number of states be N_s. S_t takes values s_i where $i \in \{0, 1, ..., N_s\}$. Then the Markov property is given by

$$P(S_t|S_{t-1}, S_{t-2}, ..., S_0) = P(S_t|S_{t-1}) \ . \tag{2.1}$$

Processes that possess the Markov property are *Markov processes* which can be described via *Markov models*. Two commonly used Markov models are the *Markov chain* and the *hidden Markov model*. The difference between the two being whether the state transitions can be directly observed from the outcomes of the random process. The two models are described in detail in the next sections.

Markov models are used prolifically in stochastic modeling. In the fields of finance, genetics, information theory, and economics they are widely used to model systems such as stock price fluctuations, bacteria reproduction, queuing theory, etc.

2.1.1 Markov Chains

Markov chains can be described by finite state machines. In Fig. 2.2, a state diagram of a Markov model is shown.

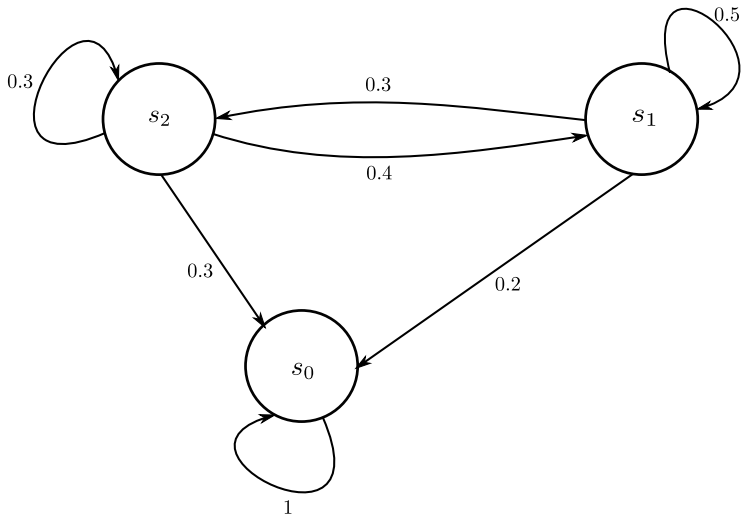


Figure 2.2: A discrete-time Markov chain

When the state transitions of a system are fully observable form the outcomes, the Markov model is called a Markov chain. Thus, Markov chains are sequences of random variables which describe a sequence of states of a stochastic process. The transition probabilities of a Markov model are given by the transition probability matrix. For this thesis, we only consider Markov chains on a finite state space with a discrete time index. Such chains are called Discrete-time Markov chains (DTMC). When the time index of a process is continuous, they are called continuous-time Markov chains (CTMC).

A **transition matrix** $\mathbf{Q}$ is a matrix used to describe a Markov model, given in Eq. (2.2), where the $(i,j)^{th}$ entry of the matrix provides the probability to transition from the i^{th} state to the j^{th} state. The transition matrix corresponding to Fig. 2.2 is shown below.

$$\mathbf{Q} = \begin{bmatrix} P(s_0|s_0) & P(s_1|s_0) & P(s_2|s_0) \\ P(s_0|s_1) & P(s_1|s_1) & P(s_2|s_1) \\ P(s_0|s_2) & P(s_1|s_2) & P(s_2|s_2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.2 & 0.5 & 0.3 \\ 0.3 & 0.4 & 0.3 \end{bmatrix}. \tag{2.2}$$

Each row of the matrix sums up to 1 since this is a stochastic model. The transition matrix given here describes a time-homogeneous Markov chain.

A **time-homogeneous Markov chain** has the property that the transition matrix is time-invariant, i.e., the transition probabilities do not change with the time steps that have elapsed.

Transition matrices provide the probabilities to be at different states at a future time step t if the matrix is multiplied t times. This means that the $t-$ step transition matrix, $\mathbf{Q}^t$ has the

following decomposition,

$$\mathbf{Q}^t = \underbrace{\mathbf{Q}\cdot\mathbf{Q}\cdot ...\mathbf{Q}}_{t-\text{times}} = \begin{bmatrix} P(S_t = s_0|S_0 = s_0) & P(S_t = s_1|S_0 = s_0) & P(S_t = s_2|S_0 = s_0) \\ P(S_t = s_0|S_0 = s_1) & P(S_t = s_1|S_0 = s_1) & P(S_t = s_2|S_0 = s_1) \\ P(S_t = s_0|S_0 = s_2) & P(S_t = s_1|S_0 = s_2) & P(S_t = s_2|S_0 = s_2) \end{bmatrix} . \tag{2.3}$$

Let a state distribution vector at time t be denoted by π_t. If at time $t-1$ the process is at state s_1, then

$$\pi_t = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{Q} \, , \tag{2.4}$$

$$= \begin{bmatrix} P(s_0|s_1) & P(s_1|s_1) & P(s_2|s_1) \end{bmatrix} \, . \tag{2.5}$$

The **steady-state distribution** of a Markov chain is the stationary probability distribution of the states to which the model converges after many time steps have elapsed. It is denoted by π. Since it is independent of the time elapsed,

$$\pi \cdot \mathbf{Q} = \pi \, . \tag{2.6}$$

The stationary state distribution can be obtained from the eigenvalue and eigenvectors of $\mathbf{Q}$.

$$\begin{aligned} (\pi \cdot \mathbf{Q})^T &= \pi^T \\ \mathbf{Q}^T \cdot \pi^T &= \pi^T \\ \left(I - \mathbf{Q}^T\right) \pi^T &= 0 \, . \end{aligned} \tag{2.7}$$

From Eq. (2.7), the stationary state distribution π is given by the left eigenvector of the transposed transition matrix associated to eigenvalue 1.

A process having Markov property is often referred to as 'memory-less', since the next step in time is independent of the past. However, the current state influences the next step, so there is a dependency or 'memory' present between two subsequent time steps. As mentioned in the introduction, the goal of this thesis is to investigate dependencies present in different parts of our transmission system model. This dependency is referred to as memory in here.

While Markov models are generally defined to possess the Markov property, it is possible to introduce Markov models with higher order memory. The standard Markov model having the Markov property is commonly termed the order-1 Markov model and analogously models with dependency arising from the two immediate previous states are termed order-2 Markov models, and so forth. In this manner, it is possible to incorporate memory or dependency from

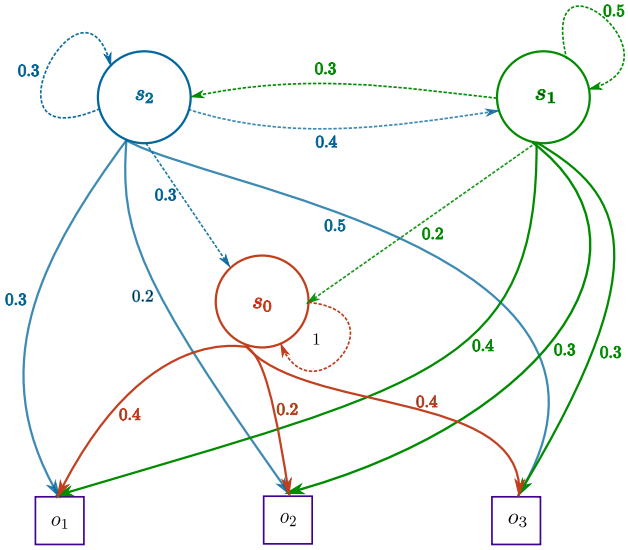


Figure 2.3: A hidden Markov Model

arbitrary number of past states, thus making them powerful stochastic modeling tools.

In the remainder of the thesis we have used the term Markov model to refer to a discrete-time Markov chain. We have used both order-1 and order-2 models.

2.1.2 Hidden Markov Models

A hidden Markov model (HMM) is one in which the states of the system are not readily observable from the outcomes. In an HMM, we observe a sequence of outcomes which are dependent on an underlying state space, although the state space is not observable.

Let the outcomes of a Markov process be denoted by the random variable O, and let the states be denoted by S. Then according to the Markov property,

$$P(O_t|S_t, S_{t-1}, ..., S_0, O_{t-1}, ..., O_0) = P(O_t|S_t) \ . \tag{2.8}$$

In Fig. 2.3, we observe that the outcomes o_i are dependent on the states s_i. By observing a sequence of outcomes o_t, we cannot immediately deduce which states the model transitioned through. For hidden Markov models, there is an additional probability matrix describing the probability of the outcomes dependent on the states, called the emission probability matrix,

denoted here by $\mathbf{E}$.

$$\mathbf{E} = \begin{bmatrix} P(o_1|s_0) & P(o_2|s_0) & P(o_3|s_0) \\ P(o_1|s_1) & P(o_2|s_1) & P(o_3|s_1) \\ P(o_1|s_2) & P(o_2|s_2) & P(o_3|s_2) \end{bmatrix} = \begin{bmatrix} 0.4 & 0.2 & 0.4 \\ 0.4 & 0.3 & 0.3 \\ 0.3 & 0.2 & 0.5 \end{bmatrix}. \tag{2.9}$$

An HMM can thus be parameterized by $\Theta = (\mathbf{Q}, \mathbf{E}, \pi_0)$.

For Markov chains, since the outcomes of the process directly relate to the states (it can be assumed that the outcomes are the states), it is quite straight-forward to generate sequences given a starting distribution or to compute the probabilities of different sequences. For a detailed description on Markov models, please refer to [2]. For a tutorial like treatment of HMMs, please refer to [3].

For hidden Markov models, there are three kinds of problems which can be formulated [2]:

- **Likelihood computation**: Given an observation sequence $\mathbf{o} = \{O_1, O_2, ..., O_T\}$ and an HMM Θ, what is the probability of the observation sequence given the model, $P(O|\Theta)$?
- **Decoding**: Given an observation sequence $\mathbf{o} = \{O_1, O_2, ..., O_T\}$ and an HMM Θ, what is the most likely state sequence $\mathbf{s} = \{S_0, S_1, ..., S_t\}$ of the model?
- **Training**: For a given observation sequence $\mathbf{o} = \{O_1, O_2, ..., O_T\}$ and the set of states $\mathbf{s}$, how do we find the parameters of the model: $\mathbf{Q}, \mathbf{E}$?

In this thesis, we are interested in the decoding problem. In order to find the most likely sequence of states which generated a particular sequence of observations for a specific model, two commonly used algorithms are the Viterbi and the BCJR algorithms. Both are trellis-based methods.

2.1.3 Trellis Decoders

In Fig. 2.4, a trellis is shown. Each segment of a trellis diagram shows the possible state transitions of a finite state-machine. In each segment, all states are represented and the transition from one state to possible other states are represented through **branches**. By concatenating T segments, it is possible to trace all possible sequences of states of length-T which could have been generated by the finite state machine.

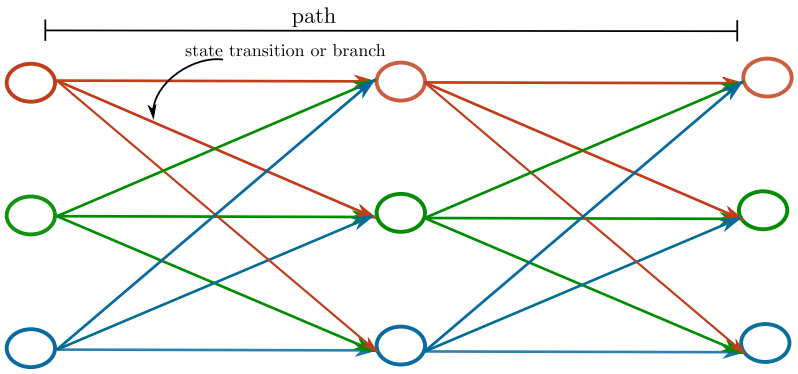


Figure 2.4: A two-segment trellis for a 3-state system

In trellis based decoding, a branch metric is computed for all branches for an observation sequence. This allows for the computation of **path metrics** which are due to the branches along the specified path.

A branch metric $\lambda_t(s', s)$ is defined as the metric from state s' at time $t-1$ to state s at time t for an observation o_t. For an HMM, the branch metric is

$$\lambda_t(s', s) = P(s|s') \cdot P(o_t|s)\,. \tag{2.10}$$

2.1.3.1 Viterbi Decoding

The Viterbi algorithm, proposed as a decoding algorithm for convolutional codes by Andrew J. Viterbi in [4] is a dynamic programming algorithm for an HMM process. As noted in [3], the Viterbi algorithm has been independently invented in many diverse fields such as information theory, molecular biology, and computer science by different authors. For a tutorial overview of its derivation and uses, [5] is an excellent resource.

The Viterbi algorithm is a sequence estimator, i.e., it decodes to a state sequence given an observation sequence and an automaton $\mathbf{\Theta}$. Of the possible $2^{N_s \cdot T}$ sequences, it decodes to one which minimizes deviation between the observed data and the data which would have been produced by the chosen state sequence.

In the Viterbi algorithm, for decoding HMMs, at the starting step, branch metrics are computed for a given starting state distribution π_0.

For every subsequent time index at all states, a cumulative metric is computed, by adding the branch metric to the path metrics on valid transitions. At every state, the path with the best metric is stored. The algorithm progressively traces out paths in the trellis, one step at

a time, which best fits the observation sequence. At every time step, at every state, only the path with the cumulative best metric is selected and stored. Since it is a sequence estimator, a cumulative metric $\Gamma_t(s')$ is used which quantifies the metric upto time t for every state s'.

The complexity of Viterbi decoding is $\mathcal{O}(2^{N_s})$ for N_s states, this is computationally much simpler than processing all $2^{N_s T}$ paths [4].

In Alg. 1, the Viterbi algorithm for sequence estimation is provided.

Algorithm 1: The Viterbi Algorithm

1 **Variables** :-
2 $\lambda_t(s', s)$: Branch Metric;
3 $\Gamma_{t-1}(s')$: Surviving path metric to state s' at time $t-1$;
4 $\Gamma_t(s', s)$: Cumulative metrics for the path segments connecting s' to s at time t;

5 **Initialize** : $\Gamma_0(s') = 0$;

6 **for** *$t = 1$ to T:* **do**
7 Compute all possible branch metrics $\lambda_t(s', s)$;
8 Compute $\Gamma_t(s', s)$: Compute the cumulative metrics for all state transitions from s' to $s$from time $t-1$ to t ;
9 `Compare and Select(` $\Gamma_t(s) = best(\Gamma_t(s', s))$`.)` For each state s, compare the incoming path metrics at time t and select and store only the path with the best metric in $\Gamma_t(s)$.
10 **end**

Result: Output the path which leads to the best metric at the end time step.

The branch metric was not defined explicitly in the algorithm provided in Alg. 1. It is computed according to the data available. For HMMs,

$$\Gamma_t(s', s) = \Gamma_{t-1}(s') \cdot \lambda_t(s', s) \; . \tag{2.11}$$

For HMMs, the starting state could be given by an initial state distribution π_o or it could be taken as the steady-state distribution π. For convolutional codes, the starting state is often the all-zero state, since the encoder also starts from the all-zero state.

In Alg.1, line number (9), the select operation is to select the best metric. For HMMs, the 'best metric' refers to the path with the highest probability.

Variants of the Viterbi algorithm include log-domain Viterbi and the soft-output Viterbi algorithm (SOVA). The log-domain Viterbi is more stable numerically and has the advantage that the multiplication of Eq. (2.11) is replaced by an addition.

2.2 Channel Codes

Channel coding, i.e., error correction coding is a method through which information is augmented using redundant or **parity** bits such that errors altering the information vector can be corrected. This provides protection against noise. There are many different channel codes available, of these we use LDPC codes in this thesis as our chosen method.

Linear codes

A linear code, $\mathcal{C}$, consist of codewords $\boldsymbol{c}_i$ such that

$$a \cdot \boldsymbol{c}_i + b \cdot \boldsymbol{c}_j \in \mathcal{C} \ , \quad a, b \in \mathbb{F}_q. \tag{2.12}$$

when the codeword elements are from the field $\mathbb{F}_q$. A code $\mathcal{C}$ with parameters (N, K) are a list of vectors of length N which have been augmented by $N - K = M$ parity bits from information vectors of length K. The **code rate** $R = K/N = \frac{\log_2 |C|}{N}$ quantifies how much information is contained in the codewords. In mathematical terms, a linear code $\mathcal{C}$ with parameters (N, K) is a $K-$ dimensional subspace of the vector space of $\mathbb{F}_q^N$ with elements from the finite filed $\mathbb{F}_q$. The subspace has dimension K, hence there are K basis vectors spanning the vector space. This set of basis vectors are the information vectors, $\boldsymbol{i}$ of the code $\mathcal{C}$. There are then q^K information vectors and they are transformed via a linear map to q^K codewords. The total number of vectors in the space are q^N. We use $GF(2)$ as the underlying field in this thesis, thus the number of valid codewords in an (N, K) linear code is 2^K.

Thus, channel codes map information vectors to codeword vectors in a one-to-one fashion. The codewords are then transmitted (or stored) and may be corrupted by noise and be converted into any one of the q^N vectors in that space. The decoding task is to deduce from the received vector which one of the information vectors was encoded.

In order to decode efficiently, we introduce a few important concepts first.

Distance Metrics

Different distance metrics are useful for different channels. For digital channels like the Binary Symmetric Channel (BSC), we use a bit-wise distance between the components of the codeword and the received word, this metric is the **Hamming distance**, d_H. The Hamming distance between two vectors **a** and **b** of length n with components a_i and b_i that may be elements of an arbitrary number field, are given as the number of components that differ from each other

$$d_H = |M| \,, \quad M = \{j : a_j \neq b_j\} \,. \tag{2.13}$$

For transmission over analog channels, such as the Additive White Gaussian Channel (AWGN), we use an analog measure, the Euclidean distance. The Euclidean distance measures the distance between a received analog value y and a transmitted value $x \in \{\pm 1\}$ for binary transmission. The probability of y to be received for a transmitted value x uses the Euclidean distance, $d_E = \sqrt{(y-x)^2} = |y-x|$ as

$$P(y|x = \pm 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{|y-x|^2}{2\sigma^2}} \;; \tag{2.14}$$

where the noise process has parameters $\mathcal{N}(\mu, \sigma^2)$. The derivation of the two different distance metrics for the two different channels are derived further on in this section, when discussing the maximum-likelihood detection criterion.

A **systematic code** is a code where the information vectors $\mathbf{i}$ can be directly read from the codeword.

Since linear codes are based on linear sub-spaces, their construction can be explained via matrices. To this end we define two matrices, the generator matrix $\mathbf{G}$ and the parity-check matrix $\mathbf{H}$.

The generator matrix $\mathbf{G}$ is the linear map which produces the codewords from the information vectors. It is a $K \times N$ matrix such that

$$\mathbf{c} = \mathbf{iG} \,. \tag{2.15}$$

The generator matrix can appear in a number of equivalent forms, elementary row operations such as Gaussian elimination do not alter the matrix, hence does not alter the code. When in standard form, the identity part of the $\mathbf{G}$ matrix ensures that the resulting codewords are systematic.

Complementary (dual) to the generator matrix in function, is the parity-check matrix $\mathbf{H}$, an $(N-K) \times N$ matrix that fulfills the following equation for all codewords $\mathbf{c}$ from a code $\mathcal{C}$.

$$\mathbf{H} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{pmatrix} = \mathbf{H}\mathbf{c}^{\mathbf{T}} = \mathbf{0} \ . \tag{2.16}$$

The parity-check matrix checks whether a codeword fulfills the $M = N - K$ parities of the code via the $M = N - K$ equations it applies to the codeword. A linear code $\mathcal{C}$ can hence also be described as the solution space of a set of parity-check equations given by the parity-check matrix $\mathbf{H}$. Only valid codewords fulfill all parity-check equations. The $\mathbf{H}$ matrix is the null-space of the code $\mathcal{C}$. $\mathbf{G}$ and $\mathbf{H}$ are related via

$$\mathbf{G}\mathbf{H}^{\mathbf{T}} = \mathbf{0} = \mathbf{H}\mathbf{G}^{\mathbf{T}} \ . \tag{2.17}$$

2.2.1 Low-Density Parity-Check (LDPC) codes

LDPC codes are a class of linear error correcting block codes characterized by sparse parity-check matrices. They were first developed in 1962 by Robert G. Gallager in his doctoral dissertation [6, 7]. In the 1960s, implementation of LDPC codes was impractical and hence they were not studied widely. With the advent of implementable iterative decoding techniques and Turbo codes in the 1990s, LDPC codes experienced a resurrection. They were also independently discovered in 1995 in [8, 9]. Since the mid 1990s, extensive research into LDPC codes has proven them to be extremely well suited for widespread implementation.

LDPC codes are capacity approaching codes, i.e., their performance is close to the theoretical limit derived from channel capacity, for a variety of channel including the Additive White Gaussian Noise (AWGN) channel. The decoding complexity also increases linearly with the block length, hence they are easily scalable.

LDPC codes have emerged as the code of choice for many applications. They are part of the digital video broadcasting version-2 (DVB-2) standard, as well as part of the Wi-Fi 802.11, IEEE 802.16 standards, and used for 10GBase-T Ethernet, LTE etc, 5G and deep space communications.

As the name suggests, low-density parity-check codes have $\mathbf{H}$-matrices which have a low

density of 1s, they are sparse. This type of construction, coupled with efficient decoding algorithms provide LDPC codes with their superior performance compared with other types of codes. The decoding algorithms can be parallelized, providing high throughput. LDPC codes can be visualized by Tanner graphs [10]. Tanner graphs are bi-partite graphs which illustrate the code structure. In Fig. 2.5, a Tanner graph for the parity-check matrix from Eq. (2.18) is shown.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} . \tag{2.18}$$

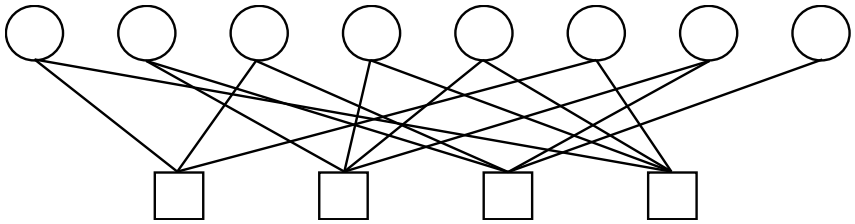


Figure 2.5: Tanner graph of $\mathbf{H}$-matrix shown in Eq. (2.18)

The circular nodes illustrate the bits of the codeword and are termed **variable nodes** (VN). The square nodes represent the parity-check equations of the code and are termed **check nodes** (CN). The degree of a variable (check) node is the number of edges or connections it has to check (variable) nodes. The connection profile between the nodes is thus a visualization of all the parity-check equations or the $\mathbf{H}$-matrix itself.

In his thesis, Gallager described codes of regular graphs, or regular connection profiles. For a code, when all the variable nodes and all the check nodes have identical degree, the code is called a regular code. In [11], Luby et al. showed that irregular codes have better error correction performance than regular codes. Irregular codes are characterized by degree distribution polynomials. The degree distribution polynomials are of the node or edge type, meaning they are probability mass functions of different degree nodes or probability mass functions of edges belonging to nodes of certain degree.

The variable and check node edge degree distribution polynomials, $\lambda(x)$ and $\rho(x)$ are

$$\lambda(x) = \sum_{i \geq 2}^{d_{\text{vmax}}} \lambda_i x^{i-1} , \tag{2.19}$$

and

$$\rho(x) = \sum_{j\geq 2}^{d_{\text{cmax}}} \rho_j x^{j-1} \,, \tag{2.20}$$

where λ_i and ρ_j are the proportions of edges connected to VNs and CNs of degree i and j, and d_{vmax} and d_{cmax} are the maximum variable and check node degrees, respectively. Similarly, the node degree polynomials, $\tilde{\lambda}(x)$ and $\tilde{\rho}(x)$ are

$$\tilde{\lambda}(x) = \sum_{i=2}^{d_{vmax}} \tilde{\lambda}_i x^{i-1} \,, \tag{2.21}$$

and

$$\tilde{\rho}(x) = \sum_{j=2}^{d_{cmax}} \tilde{\rho}_j x^{j-1} \,, \tag{2.22}$$

where $\tilde{\lambda}_i$ and $\tilde{\rho}_j$ are the proportions of VNs and CNs of degree i and j, respectively. The degrees of the nodes are from 2 upwards to avoid trivial cases. The conversion from one parameterization to the other is given by,

$$\tilde{\lambda}_i = \frac{\lambda_i/i}{\sum_k \lambda_k/k} \,, \quad \tilde{\rho}_j = \frac{\rho_j/j}{\sum_k \rho_k/k} \,, \tag{2.23}$$

$$\lambda_i = \frac{i\tilde{\lambda}_i}{\sum_k k\tilde{\lambda}_k} \,, \quad \rho_j = \frac{j\tilde{\rho}_j}{\sum_k k\tilde{\rho}_k} \,. \tag{2.24}$$

The rate of a code, R can be derived from the degree distribution polynomials as

$$R = 1 - \frac{\sum_{j\geq 2}^{d_{\text{cmax}}} \frac{\rho_j}{j}}{\sum_{i\geq 2}^{d_{\text{vmax}}} \frac{\lambda_i}{i}} \,. \tag{2.25}$$

We define a few important properties of LDPC codes in the following.

A **cycle** is a path along the edges of the Tanner graph where the beginning and ending node is the same. The length of a cycle is the number of edges it traverses.

The **girth** of a code is the length of its shortest cycle. Cycles of a code have an influence on the decoding performance of the code.

Cycles in the LDPC code graph degrade the performance of the iterative decoder. Finite length codes contain cycles due to the construction of the code and thus have an effect on the decoding performance. Hence, the length of the shortest cycle or the girth of a code

is an important parameter for assessing its decoding performance. The girth and cycles are addressed when discussing the decoding and $\mathbf{H}$-matrix design in later sub-sections.

Multi-edge type (MET) code

MET codes refer to codes which have multiple types of edges belonging to multiple classes of nodes. If, for example, a transmission system consists of different constituent channels, i.e., different bits of the same codeword are sent over different channels, the resulting code should have different parity-check properties to reflect the different channels. This gives rise to multi-edge type codes.

We take the following description for multi-edge type codes from [12], the degree polynomials associated with variable and check nodes, respectively for MET codes are:

$$\nu(\mathbf{r}, \mathbf{a}) = \sum \nu_{\mathbf{b},\mathbf{d}} \mathbf{r}^{\mathbf{b}} \mathbf{a}^{\mathbf{d}} \,, \tag{2.26}$$

$$\mu(\mathbf{a}) = \sum \mu_{\mathbf{d}} \mathbf{a}^{\mathbf{d}}. \tag{2.27}$$

We define the parameters $\mathbf{b}, \mathbf{d}, \mathbf{r},$ and $\mathbf{a}$ as follows. Let there be m_e edge types. The factor $\mathbf{a}^{\mathbf{d}} = \prod_{i=1}^{m_e} a_i^{d_i}$ is associated with every node in the graph and denotes the number of edge types, $\mathbf{a} = (a_1, \ldots, a_{m_e})$, that are connected to it and the associated degrees, $\mathbf{d} = (d_1, \ldots, d_{m_e})$. Let, there be m_r channels of differing parameters in the system. The factor $\mathbf{r}^{\mathbf{b}} = \prod_{i=0}^{m_r} r_i^{b_i}$ is also associated to every variable node and pertains to the received information over the channel, with $\mathbf{r} = (r_0, \ldots, r_{m_r})$ denoting the different received distributions, and the $\mathbf{b} = (b_0, \ldots, b_{m_r})$ denoting the number of connections to the different received distributions. We assume that each variable node is associated to only one channel, and thus, only one entry in $\mathbf{b}$ is set to 1 and the rest to 0.

2.2.2 Decoding of LDPC Codes

Due to the sparse nature of the $\mathbf{H}$-matrix in an LDPC code, iterative decoding methods perform extremely well. The sparse nature of the code means that a few variable nodes, or codeword symbols, participate in a single parity check. In iterative decoding, the check equations of the code are fulfilled at different iteration cycles, made possible due to the distributed and sparse parity check structure of the code. This overall leads to low probabilities of error, allowing LDPC codes to provide near capacity performance.

We introduce the notation now of transmitted and received vectors over a channel, used throughout the thesis. Let the transmitted sequence be $\mathbf{x}$ with individual elements of the

sequence denoted by x_i. Let the received sequence be $\mathbf{y}$ with the individual elements being denoted by y_i. The goal is to find a decoded sequence, $\hat{\mathbf{x}}$ such that decoding error is minimized. In this context, the Maximum-Likelihood (ML) estimate and the Maximum A-posteriori Probability (MAP) estimate are both used as decoding principles. These are methods used in Bayesian inference[1] to estimate parameters of probability distributions. A few related terms in this context are provided below.

The **likelihood function** of a stochastic process is defined as the probability function describing the likelihood of observed data, $\mathbf{y}$ for a given value of $\mathbf{x}$. A posterior distribution is the distribution of the random variable $\mathbf{x}$ based on observed data. The **prior distribution** of the variable $\mathbf{x}$ is termed the **a-priori** distribution.

Using Bayes' theorem, the MAP estimate is then

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} P(\mathbf{x}|\mathbf{y}) = \arg\max_{\mathbf{x}} \frac{\mathbf{P}(\mathbf{y}|\mathbf{x})\mathbf{P}(\mathbf{x})}{\mathbf{P}(\mathbf{y})} \ . \tag{2.28}$$

Since the maximization is over $\mathbf{x}$, $\hat{\mathbf{x}}$ is independent of $P(\mathbf{y})$. When all transmit sequences are equally likely, the MAP estimate Eq (2.28) reduces to

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} P(\mathbf{y}|\mathbf{x}) \ . \tag{2.29}$$

Generally, for decoding, the goal is to estimate the ML or the maximum a-posterior probability (APP) solution. MAP decoding of LDPC codes is computationally prohibitive, instead iterative algorithms are used which are described in the next sub-section. Both hard- and soft-value implementations of iterative decoding are possible. Hard value algorithms such as the bit-flipping algorithm are of limited interest since the performance of soft-value decoders far outperform hard-value decoders, theoretically by 3 dB. As a soft value measure, log-likelihood values are used.

We will now derive the distance metrics used for the AWGN and BSC channel using the ML criterion.

[1] Bayesian inference or Bayesian statistics refers to estimating the probability distribution of random variables via Bayes formula by utilizing observed data

For the BSC channel with crossover probability ϵ, the ML criterion yields

$$\begin{aligned}\hat{\mathbf{x}} &= \arg\max_{\mathbf{x}} \log P(\mathbf{y}|\mathbf{x}) \\ &= \arg\max_{\mathbf{x}} \log \prod_k P(y_k|x_k) \\ &= \arg\max_{\mathbf{x}} \sum_k \log P(y_k|x_k) \\ &= \arg\max_{\mathbf{x}} \left[d_H(\mathbf{y},\mathbf{x}) \log(\epsilon) + (N - d_H(\mathbf{y},\mathbf{x}) \log(\mathbf{1}-\epsilon))\right] \\ &= \arg\max_{\mathbf{x}} \left[d_H(\mathbf{y},\mathbf{x}) \log \frac{\epsilon}{1-\epsilon} + N \log(1-\epsilon)\right] .\end{aligned} \tag{2.30}$$

$\log \frac{\epsilon}{1-\epsilon} < 0$ and $N \log(1-\epsilon)$ is independent of $\mathbf{x}$, thus

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} d_H(\mathbf{y},\mathbf{x}) . \tag{2.31}$$

For the AWGN channel, Eq.(2.30) is decomposed into

$$\begin{aligned}\hat{\mathbf{x}} &= \arg\max_{\mathbf{x}} \sum_k \log \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left[-(y_k - x_k)^2/(2\sigma^2)\right] \right) \\ &= \arg\min_{\mathbf{x}} \sum_i (y_k - x_k)^2 \\ &= \arg\min_{\mathbf{x}} d_E(\mathbf{y},\mathbf{x}) ,\end{aligned}$$

where $d_E(\mathbf{y},\mathbf{x}) = \sqrt{\sum_k (y_k - x_k)^2}$. Since $\sqrt{\sum_k (y_k - x_k)^2}$ is non-negative and the square-root function is monotonous for non-negative arguments, $d_E^2(\cdot)$ has been replaced by $d_E(\cdot)$ in the last line. Thus, ML decoding for different channels leads to the use of the appropriate distance metric for the channel.

Log-likelihood Ratios

For soft-decoding, probability values or Log-likelihood Ratios (LLRs) are used. In this thesis, we use antipodal signaling, i.e., the mapping from codeword bit values c_k to transmit values x_k is given by

$$x_k = (-1)^{c_k} \; ; \;\; c_k = \{0,1\} . \tag{2.32}$$

An LLR for a bit x_k is then defined as

$$L(x_k) = \ln \frac{P(x_k = +1)}{P(x_k = -1)} . \tag{2.33}$$

An LLR value of the form shown in Eq. (2.33) signals a higher probability for $x_k = +1$ when $L(x_k)$ is positive and for $x_k = -1$ when $L(x_k)$ is negative. The magnitude of the value signifies the reliability. For the BEC, the LLR definition is more straightforward and intuitive. Since on the BEC there are no errors possible, the LLRs corresponding to $\{+1, -1\}$ are respectively, $\{+\infty, -\infty\}$ and for an erasure it is zero. We will use L to denote LLR values in the following.

2.2.2.1 Belief-propagation Decoding

There have been a few iterative decoding algorithms presented in literature for LDPC codes. They can be classified according to a discrete message alphabet: Gallager A and B, or a continuous message alphabet: Belief-propagation [13]. We present the Belief propagation (BP) algorithm in the following.

BP decoding is a message passing decoder. Message passing decoders are a type of Bayesian inference decoder which work in a parallel and distributed fashion, on every bit of the codeword. In Belief-propagation (BP) decoding, 'beliefs' (probabilities or LLRs) are propagated through a network (Tanner Graph) to obtain information on certain variables (variable and check nodes) within the network. A general discussion of BP decoders is given in [14].

At the variable nodes, the algorithm computes the following. We now consider a code vector $\mathbf{x}$ being transmitted and determine the logarithmic APP ratio $L(x_k|\mathbf{y})$ for the received word $\mathbf{y}$.

$$\begin{aligned} L(x_k|\mathbf{y}) &= \ln \frac{P(x_k = +1|\mathbf{y})}{P(x_k = -1|\mathbf{y})} \\ &= \ln \frac{P(\mathbf{y}|x_k = +1)}{P(\mathbf{y}|x_k = -1)} + \ln \frac{P(x_k = +1)}{P(x_k = -1)} \\ &= \underbrace{\ln \frac{P(y_k|x_k = +1)}{P(y_k|x_k = -1)}}_{\text{intrinsic}} + \underbrace{\ln \frac{P(\mathbf{y}_{\setminus k}|x_k = +1)}{P(\mathbf{y}_{\setminus k}|x_k = -1)}}_{\text{extrinsic}} + \underbrace{\ln \frac{P(x_k = +1)}{P(x_k = -1)}}_{\text{a priori}}. \end{aligned} \tag{2.34}$$

The received vector $\mathbf{y}$ can be partitioned into y_k and $\mathbf{y}_{\setminus k}$ ($\mathbf{y}$ without the kth component) terms due to a memory-less channel assumption (independence). The extrinsic information is labeled $L_E(x_k|\mathbf{y}_{\setminus k})$. The a-priori information is labeled $L_A(x_k)$, and the intrinsic LLR is labeled $L_{\text{itr}}(x_k)$.

The three types of information extrinsic, a-priori, and intrinsic are independent measures (when there are no cycles in the graph), hence the associated probabilities are multiplied, or LLRs

added.

A-priori Information

Relates to the a-priori distribution of the source data sequence.

$$L_A(x_k) = \ln \frac{P(x_k = +1)}{P(x_k = -1)} \, . \tag{2.35}$$

Intrinsic Information

Relates to the channel properties. For transmission over an AWGN channel, the transmitted sequence is affected by zero-mean Gaussian noise samples, resulting in analog received values

$$y_k = x_k + z_k \, . \tag{2.36}$$

where z_k are independent and identically distributed (i.i.d.) noise samples from the Gaussian distribution $\mathcal{Z}(0, \sigma_n^2)$ where σ_n^2 is the variance of the zero-mean noise process. The intrinsic LLR is then calculated by

$$\begin{aligned} L_{\text{itr}}(x_k) =& \ln \frac{P(y_k|x_k = +1)}{P(y_k|x_k = -1)} \\ =& \ln \frac{\frac{1}{\sqrt{2\pi}\sigma_n} \cdot e^{-(y-1)^2/2\sigma_n^2}}{\frac{1}{\sqrt{2\pi}\sigma_n} \cdot e^{-(y+1)^2/2\sigma_n^2}} \\ =& \frac{2y}{\sigma_n^2} \, . \end{aligned} \tag{2.37}$$

Extrinsic Information

Relates to the code constraints, i.e., the parity checks. The extrinsic and intrinsic information together constitute the ML estimate.

The message update equations during BP decoding are presented below. Each bit of each codeword can be decoded simultaneously, since belief propagation is a fully parallelizable method. The LLR messages are passed along the edges of the Tanner graph, the outgoing message at the l^{th} iteration from variable node v to check node w is given by

$$L_{v \to w}^{(l)} = L_A(v) + L_{\text{itr}}(v) + \sum_{w'} L_{w' \to v}^{(l-1)} \quad , \quad w' \in \{\{\mathcal{N}_v\} - \{w\}\}, \tag{2.38}$$

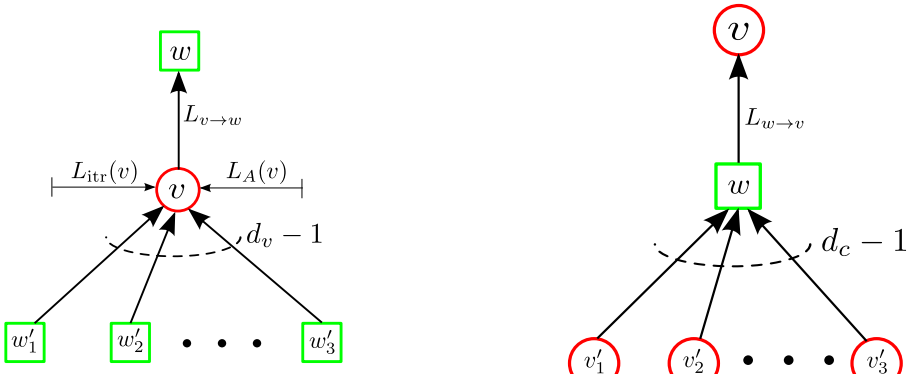
where $\{\mathcal{N}_v\}$ is the set of neighbors of VN v.

The outgoing message from check node w to variable node v at the l^{th} iteration is given by

$$\tanh \frac{L_{w\to v}^{(l)}}{2} = \prod_{v'} \tanh \frac{L_{v'\to w}^{(l)}}{2} \quad , \quad v' \in \{\{\mathcal{N}_w\} - \{v\}\}, \tag{2.39}$$

where $\{\mathcal{N}_w\}$ is the set of neighbors of CN w.

In the update equations provided in eqs. (2.38) and (2.39), the LLRs on an outgoing edge were computed by excluding the LLR incoming on that edge. This is reflective of the formulation $L_E(x_k|\mathbf{y}_{\setminus k})$. In order to continue to iteratively pass messages successfully, it is imperative to not 'recycle messages', otherwise the MAP structure of Eq. (2.34) would be violated. The updates are illustrated in figures 2.6a and Fig 2.6b.



(a) Update of variable nodes.

(b) Update of check nodes.

Figure 2.6: Node operations of the message-passing algorithm

The BP decoding algorithm is almost optimal. The algorithm delivers a-posteriori probability estimates, but is not a MAP decoder since there are cycles present in the graph. The girth of a code influences information recycling inside the graph and hence degrades the code performance. Specifically, for a graph with girth g, the messages start to be recycled after $g/2$ iterations. The BP algorithm is also commonly referred to as the sum-product algorithm (SPA), the algorithm is provided in Alg. 2.

In MAP or ML decoding, the minimum distance of the code determines the error floor of the code. For iterative message passing decoders, certain subsets of the Tanner graph induce subgraphs which lead to error floors. To this end, we now define **trapping-sets**. In the high SNR region, BP decoder performance can suffer from an error floor, which is due to trapping-sets. Conceptually related to trapping sets are stopping sets, pseudocodewords, near-codewords, absorbing sets etc.

Algorithm 2: The Sum-Product Algorithm

Variables :

$L_A(v)$: a-priori information for VN v
$L_{\text{itr}}(v)$: intrinsic information for VN v
$V = \{v_1, ..., v_N\}$: set of VNs
$W = \{w_1, ..., w_M\}$: set of CNs
$E_v = \{e : H(e, v) = 1\}$: set of CNs connected to VN v
$E_w = \{e : H(w, e) = 1\}$: set of VNs connected to CN w
L_{v-w_e}: message from VN v to CN w_e
L_{w-v_e}: message from CN w to VN v_e
$\hat{V} = \{\hat{v_1}, ..., \hat{v_N}\}$: decoded codeword ;

Initialize : $\forall v \in V$

$L_A(v)$: according to Eq. (2.35)
$L_{\text{itr}}(v)$: according to Eq. (2.37) ;

for *l = 1 to L:* **do**

  VN Update:

  $L^{(l)}_{v-w_e} = L_A(v) + L_{\text{itr}}(v) + \sum_{e':E_v-\{e\}} L^{(l-1)}_{w_{e'-v}} \ , \quad \forall e \in E_v \ , \forall v \in V$

  CN Update:

  $\tanh \frac{L^{(l)}_{w-v_e}}{2} = \prod_{e':E_w-\{e\}} \tanh \frac{L^{(l)}_{v'_e-w}}{2} \ , \quad \forall e \in E_w \ , \forall w \in W$

  Intermediate decoding result at iteration l:

  $L^{(l)}_v = L_A(v) + L_{\text{itr}}(v) + \sum_{E_V} L^{(l-1)}_{w_{e-v}}, \ \ , \forall v \in V$

  $$\hat{v} := \begin{cases} 0 : L^l_v > 0 \\ 1 : L^l_v < 0 \end{cases} \ , \forall v \in V$$

  if $\mathbf{H}\hat{V}^T = 0$ **then**

    break for loop

  end

end

Result: Output $\hat{V}$

In [15], 'near-codewords' are introduced to explain the error-floor region. In [16], the terminology trapping sets is introduced instead of 'near-codewords', since such sets are a function of the decoding algorithm and give rise to vectors which have a non-zero syndrome. We define near-codewords or trapping-sets analogous to [17]. Let there be a non-empty set of variable nodes, $\mathbf{T}(\mathbf{y})$, which are not 'eventually correct', i.e., these bits of the codeword cannot be corrected regardless of the number of iterations. Then $\mathbf{T}(\mathbf{y})$ is an (a, b) trapping set where $a = |\mathbf{T}(\mathbf{y})|$ and b is the number of odd-degree check nodes in the induced sub-graph of $\mathbf{T}(\mathbf{y})$. 'Near-codewords' having small values of a and relatively small b lead to errors that the SPA decoder cannot correct. This can be explained by considering the all-zero codeword to be transmitted [15]. Low values of a imply a low weight error pattern, which are more probable on average than higher weight error patterns. Additionally, low values of b imply that only a small number of check equations are affected. Since only a few of the parity-check nodes are affected, the distributed (over the entire graph) correction which lends LDPC decoders their superior performance fails, the decoder cannot escape these error states.

Iterative decoding of LDPC codes can be explained using *computation trees* [18] and how the SPA performs on computation trees [19]. In [20], the same authors of [19] provide an analysis of the error floor due to pseudo-codewords using a signal space analysis. They show that unlike MAP or ML decoding where the decoder decodes to a codeword which is closest to the channel output in Euclidean distance, in BP decoding the decoded codeword is the signal with the highest correlation to the channel output. Due to the nature of BP on a graph with cycles, there are codewords on the computation tree (tree code) which have a higher correlation to the channel output. They are not codewords, but the algorithm decodes to them due to the higher correlation, these are termed pseudo-codewords. False correction to pseudo-codewords is a phenomenon of cycles in the Tanner graph. In [21], iterative decoding and error floors are studied from the perspective of graph-covers, where the computation trees of two graphs may be topologically equivalent, even if the graphs are not. Since BP decoding is a locally operating algorithm on the computation tree, it considers all solutions of all finite-length covers of the Tanner graph and can thus lead to decoding failure. This paper also shows the equivalence between BP decoding and linear-programming (LP) decoding. Absorbing Sets are studied in [22].

Stopping sets can be considered a special subclass of trapping sets, for the BEC channel [16]. For a BEC channel, the set of erasures which stops decoding is known as the stopping set $\mathcal{S}$. A stopping set $\mathcal{S}$ is a set of variable nodes where all neighbors of $\mathcal{S}$ are connected to $\mathcal{S}$ at least twice. Thus, when these bits are erased, it is not possible to recover the codeword.

Stopping sets are investigated in [23, 24].

2.2.3 Optimization of LDPC Codes

The design of optimized LDPC codes is an extensively studied subject. Optimized LDPC codes are codes designed for a specific environment providing optimum performance dependent on the parameters of the system (such as the SNR or rate requirements) and the characteristics of the code (such as multi-edge-type codes). Code optimization depends on an analysis technique named **Density Evolution** (DE), first introduced in [13]. Density evolution is a technique which provides a method for computing the capacity of LDPC codes. In this section, we will describe density evolution and code optimization based on density evolution.

We start this subsection by defining a few information theoretic concepts, *Entropy*, *Mutual Information (MI)*, and *Capacity*.

Entropy

Information entropy quantifies the amount of information contained (uncertainty) in a random variable.

In 1928, Hartley quantified the information contained in a random variable as $\log_b m$, where m is the number of outcomes of a random process. However this definition of entropy does not include the probability of the individual outcomes. In 1948, Shannon defined [25] entropy of a random variable X as $H(X)$, which is negatively proportional to the probability of the outcome.

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \ , \tag{2.40}$$

measured in bits. We are omitting the base in subsequent formulations.

The conditional entropy of a random variable Y conditioned on the random variable X is given by,

$$\begin{aligned} H(Y|X) &= \sum_{x \in \mathcal{X}} P(x) H(Y|X = x) \\ &= -\sum_{x \in \mathcal{X}} P(x) \sum_{y \in \mathcal{Y}} P(y|x) \log P(y|x) \\ &= -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} P(x, y) \log P(y|x) \ . \end{aligned} \tag{2.41}$$

The conditional entropy $H(Y|X) = 0$ if the RV X completely determines the RV Y.

Mutual Information

Mutual information, measured between two random variables, quantifies the reduction in uncertainty of one random variable conditioned on knowledge of the second one, i.e., in terms of entropy of the variables. For two discrete random variables X and Y, the mutual information, denoted by $I(X;Y)$ is given by,

$$\begin{aligned} I(X;Y) =& H(X) - H(X|Y) = H(Y) - H(Y|X) \\ &= \sum_{x\in\mathcal{X}, y\in\mathcal{Y}} P_{X,Y}(x,y) \log \frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)} \ . \end{aligned} \tag{2.42}$$

When the mutual information between two random variables is zero, they are statistically independent.

Capacity

The capacity of a channel is a tight upper bound on the rate of information transmitted over a channel in the presence of interference.

$$C = \sup_{p_x} I(X;Y) \ . \tag{2.43}$$

The channel input X has average power $P = \mathbb{E}(X^2)$ and $Y = X + Z$. The noise power is given by $N_Z = \mathbb{E}(Z^2)$.

$$\begin{aligned} C &= \sup_{p_x} H(Y) - H(Y|X) \\ &= \sup_{p_x} H(Y) - H(Z) \\ &= \log_2 \sqrt{2\pi \exp(P + N_Z)} - \log_2 \sqrt{2\pi \exp(N_Z)} \\ &= \frac{1}{2} \log_2 \left(1 + \frac{P}{N_Z}\right) \ ; \end{aligned} \tag{2.44}$$

where the units are in bits/channel symbol. This formula is known as the Shannon capacity, it is the upper limit for an AWGN channels capacity. The distribution of X is chosen as Gaussian, which maximizes $H(Y)$, which in turn maximizes the capacity for the channel over all possible signal sets. When the channel symbol rate is R_s symbols/second, the capacity

becomes

$$C = \frac{R_s}{2} \log_2 \left(1 + \frac{P}{N_Z}\right) \text{ bits/second.} \tag{2.45}$$

For a channel limited to bandwidth W Hz, the maximum distortion free symbol rate is (*sampling theorem*) $R_{s,max} = 2W$, which yields

$$C = W \log_2 \left(1 + \frac{P}{N_Z}\right) \text{ bits/second.} \tag{2.46}$$

2.2.3.1 Density Evolution

BER curves of iteratively decoded LDPC codes exhibit a **threshold** behavior, above a certain threshold SNR, the error probability becomes diminishingly small as the block length tends to infinity. This behavior is observed for many different channels such as the BEC, BSC, AWGNC, etc. Thus, the threshold of a code is a parameter which can be used as a measure of the performance of a code, since it signals the convergence point of a code. The threshold of a code can be calculated via the technique of density evolution. In this subsection, we present the important DE results in code design, based on [11–13, 26].

We will first develop the concept of density evolution, based on [12, 13, 26]. In order to evaluate the performance of a message passing decoder over a channel for a code, the expected fraction of erroneous messages passed along the edges are tracked. If the fraction of error messages for a length N code, $P_e^N(l)$ goes to zero as the number of iterations l tends to infinity, then the code is converging.

In order to track the fraction of erroneous messages evolving with iterations, the message (LLR) densities along the edges are tracked. This evolution of densities allows for the computation of the code performance. Gallager had proposed a similar analysis in [7] for the BSC for regular codes, which was then extended to ensembles of codes (not a particular code) in [11] over the BEC, BSC, and for irregular codes, too. In [13], following [11], density evolution was generalized for message passing decoders over any binary-input memoryless channel over discrete or continuous alphabets. The major conclusions of the theory of DE are summarized as follows:

- **Concentration:**

  For a specific degree distribution pair (λ, ρ) and a chosen codeword length N, there are many realizations of the code possible depending on the edge connection profile of the code graph. This family of codes, $\mathcal{C}^N(\lambda, \rho)$ is termed an *ensemble*.

The fraction of error messages are computed as an expectation over all instances of the ensemble, choice of message, and realization of noise. It suffices to analyze the average performance of the ensemble, and it is not necessary to analyze the performance of individual realizations of code, channel, or message, since the particular results are concentrated around the expected behaviour. The convergence to the expected behaviour is exponential in N.

- **Convergence:**

  The expected fraction of errors $P_e^N(l)$ of an ensemble of length N converges to the asymptotic case $P_e^\infty(l)$ for large N.

- **Threshold:**

  For the asymptotic case, i.e., cycle-free graphs, the expected fraction of error messages can explicitly be computed. There is also a threshold σ^*, a channel quality parameter (like the noise variance or erasure probability) such that for channels with a better quality $\lim_{l\to\infty} P_e^\infty(l)$ goes to zero. For channels with quality parameter worse than the threshold, the error probability is a strictly positive number, bounded away from zero, independent of the iteration number or code length.

The DE algorithm is further based on three symmetry conditions, from Definition 1 of [13], channel symmetry, as well as variable- and check-node symmetry.

In [27], the authors develop a method for easier computation of the densities of messages over AWGN channels. On the BEC, the evolution of LLR densities are a one-dimensional problem, however, for the AWGN the problem is computationally more involved. A simplification of the problem can be obtained by considering a **symmetry condition** which is preserved under density evolution for all densities, which states, given $f(x)$ as a message density, $f(x) = f(-x)\exp^x$. Under the symmetry condition, called 'consistency,' Gaussian densities have the special property that the mean of the density μ and the variance σ^2 are related via $\sigma^2 = 2\mu$. Then it is sufficient to keep track of the mean (or variance) only, during the iterations, which makes the procedure tractable.

It is also possible to use the Mutual Information between the decoder output and channel input for evaluating the evolution process of the messages. A function J is defined which computes the mutual information over a channel for consistent densities, i.e., densities fulfilling the symmetry condition.

$$J(m) = 1 - E_x(\log_2(1 + e^{-x})), \quad x \sim N(m, 2m) \; . \tag{2.47}$$

J is continuous and a strictly monotonous function, so the mean (or variance) of the messages can also be computed from the mutual information.

EXIT charts

Another method for computing code performance is by the use of EXIT (extrinsic-information-transfer) charts. The technique of EXIT charts can be applied to both LDPC and Turbo codes. We discuss Exit charts for LDPC codes here. The theory follows from [28–30].

The operations at the variable nodes and check nodes constitute one full iteration of the LDPC decoder. Every half iteration, i.e., after only the variable or the check node operation, the decoding performance should improve. As the metric capturing the performance, we use mutual information. Transfer curves can be plotted for both variable and check nodes, plotting the average output mutual information against the average input mutual information. Additionally, the input MI to the check nodes is the output MI from the variable nodes, hence both curves can be shown on the same plot. By plotting the curves on the same figure with the abscissa and ordinate of one of the curves reversed, the convergence behavior of the iterative decoder can be illustrated. By following the input-output relationship between them, it can be graphically deduced whether mutual information of 1 will be achieved for a specific code, which signals convergence. The variable nodes are connected to the channel, hence for different channel parameters, different plots are obtained. The decoding threshold for different codes can thus be graphically illustrated via LDPC EXIT charts.

DE Equations for Consistent Densities:

Let $x_{cv}^{(l)}$ and $x_{vc}^{(l)}$ be the mutual information associated with messages coming from check nodes to variable nodes and from variable nodes to check nodes, respectively. The evolution of messages is then,

$$x_{vc}^{(l)} = \sum_{i=2}^{d_{vmax}} \lambda_i J(\frac{2}{\sigma^2} + (i-1)J^{-1}(x_{cv}^{(l-1)})) \tag{2.48}$$

and

$$x_{cv}^{(l)} = 1 - \sum_{j=2}^{d_{cmax}} \rho_j J((j-1)J^{-1}(1-x_{vc}^{(l)})) \ . \tag{2.49}$$

Since our interest lies in designing optimized irregular codes, we develop the steps of the theory of Density Evolution to that end. Based on the evolution equations, an optimization routine can be used to find the best codes. The details of this are provided in Chapter 4. In [11], the authors present a linear programming solution to find codes based on code design

constraints. The preferred method, and one which we also follow in this thesis, is to choose a check-node degree polynomial and find a variable-node degree polynomial that will satisfy a given rate requirement, since rate and degree polynomials are related via Eq. (2.25).

2.2.4 Parity-check Matrix Construction Algorithms

After obtaining the degree distribution polynomials $\lambda(x)$ and $\rho(x)$, the $\mathbf{H}$-matrix has to be constructed. Of the many different available construction algorithms, the Progressive Edge Growth (PEG) algorithm [31, 32] has emerged as a classic choice. The goal of a PEG construction is to add edges to the Tanner graph progressively such that every added edge has a minimal effect on the girth of a code.

In Alg. 3, the PEG algorithm is provided.

We expand on line 14 of Alg. 3 further. When expanding a tree from VN v_i, one of two possibilities may occur:

- all M CNs have not been reached from v_i, even if the depth l is increased, there will be no new elements in $\mathcal{N}_{v_i}^{l}$. In this case, an edge is placed to any node which has not been reached and thus, no cycles are created.
- all M CNs have been reached at some depth of the graph, as we keep increasing l, at some point we will encounter the scenario that at depth l not all CNs were reached $\bar{\mathcal{N}}_{v_i}^{l} \neq \emptyset$, but at depth $(l+1)$, all of them have been reached, $\bar{\mathcal{N}}_{v_i}^{l+1} = \emptyset$. In this case, an edge is placed between VN v_i and a check node reached at depth $l+1$, there by creating the largest possible cycle.

When connecting edges from VN v_i to a CN in the set $\bar{\mathcal{N}}_{v_i}^{l}$, there is a choice between CNs of different quality (different degrees) possible. There exist a few possibilities in this case. Choosing the CN which has lowest degree (if multiple such exist, choosing one randomly from those) yields graphs with check-regular distributions ($\rho(x)$ is a monomial) or graphs with concentrated check-node profile, i.e., check degree polynomials with neighboring degrees. Since there is evidence that graphs with concentrated check profiles are optimum, this is a good strategy. However, when there is also a $\rho(x)$ as well as a $\lambda(x)$, constraints can be placed such that only those nodes from the $\bar{\mathcal{N}}_{v_i}^{l}$ are chosen which do not violate the check-node degree profile. This is the version that has been implemented in this thesis.

One of the drawbacks of LDPC codes is encoding complexity. While decoding complexity scales linearly with block length, encoding complexity is N^2 [16, 33]. In order to mitigate this

Algorithm 3: The PEG Algorithm

Variables :-
$\mathcal{N}_{v_i}^{l}$: set of CNs reached by VN v_i at depth l
$\bar{\mathcal{N}}_{v_i}^{l}$: set of CNs not reached by VN v_i at depth l
d_{v_i}: maximum degree of VN v_i;

Initialize :- Each VN and CN is initialized with the number of sockets according to $\lambda(x)$ and $\rho(x)$.
Sort the variable nodes with decreasing degree;

for *i = N to 1* **do**
    for *k = 1 :* d_{v_i} **do**
        if *k == 1* **then**
            Connect the first node to a check node of lowest degree under the current graph setting.
        end
        else
            Expand a tree from the current node v_i up to depth l such that the cardinality of $\mathcal{N}_{v_i}^{l}$ stops increasing but is less than m, or $\bar{\mathcal{N}}_{v_i}^{l} \neq \emptyset$, but $\bar{\mathcal{N}}_{v_i}^{l+1} = \emptyset$. Then, connect an edge from VN v_i to a CN from $\bar{\mathcal{N}}_{v_i}^{l}$ which has the lowest degree and adheres to $\rho(x)$.
        end
    end
end

problem, the classical solution is to construct triangular $\mathbf{H}$-matrices [34].

We have constructed matrices which are upper triangular and systematic. This construction comes with the advantage that the codewords can be generated from the $\mathbf{H}$-matrix with complexity increasing linearly with the codeword length N. Since the matrix is upper triangular, the parity positions can be solved for recursively since the information part of the codeword is known. This modification comes at a very small performance loss, but is more easily scalable. Modifications to the PEG algorithm from Alg. 3 for constructing upper triangular matrices is given below, also from [31].

- The information part of the $\mathbf{H}$-matrix is constructed as per Alg. 3.
- For the parity part:
  - The first edge for any parity node is placed on the diagonal.
  - From the second edge onward, find the CNs that have not been reached at any depth or have been reached at the largest distance. From this set, find the subset of CNs that are in the allowed part of the graph, i.e., the part that preserves the triangular structure and connect to the lowest degree CN from this set.
  - A further modification named PEG-zigzag forces degree-2 VNs to be connected in a zigzag pattern, i.e., the partition of degree -2 VNs of the parity part of $\mathbf{H}$ have a bi-diagonal structure [32]. This modification allows avoiding cycles with degree-2 VNs which can degrade performance.

Another graph construction method of interest is the Approximate Cycle EMD or ACE algorithm which lowers error floors by limiting short cycles that are poorly connected to the rest of the graph [35]. A joint ACE-PEG approach is given in [36].

2.2.5 Turbo Codes

Turbo codes [37] are a class of iteratively decoded convolutional codes which also have capacity approaching performance. Parallel-concatenated recursive systematic convolutional (RSC) codes were first presented in 1993 in [37], serially concatenated Turbo codes are also a variant.

We use a Turbo decoding structure in Chapter 3 and explain the operating principle in detail there. Only a short description is provided here. For the parallel concatenated code, the same information sequence is encoded by (at least) two encoders, an interleaver is used on the information sequence for at least one encoder. Thus,the same information sequence gives rise to (almost) independent redundancy checks. At the receiver side, two decoders corresponding

to the two encoders work in conjunction to decode the information sequence. Decoding is performed iteratively, between the two decoders. For soft-value decoding using LLR values, each decoder has an input (bit-wise) LLR value and an output LLR value, the subtraction between the two terms yields the measure of information that the decoder has obtained in that round of iterations. This newly obtained (bit-wise) information is passed to the other decoder as a-priori information and the process continues till a hard decision on the bit-wise LLR values decides the final decoded word. De-interleaving is performed on the input to the decoders from the channel as well as on the information between the two decoders such that the input information and symbols to the decoders are in the correct order. For decoding, the trellis based BCJR (Bahl-Cocke-Jelinek-Raviv) algorithm is used. We now give a description of the BCJR algorithm.

2.2.5.1 BCJR Algorithm

The BCJR algorithm operates on maximizing the APP probability of each individual bit in a sequence. Like the Viterbi algorithm from Alg. 1, it also operates on trellises, but the computational complexity of BCJR is higher than the Viterbi algorithm.

The Baum-Welch or the Forward-backward algorithm, both of which are used for HMM training/learning operate similarly. The BCJR computes probabilities at every time index of the trellis diagram, based on the entire sequence. Whereas the Viterbi algorithm is a sequence estimator, i.e., it minimizes sequence error probability, the BCJR determines probabilities at individual time indices for individual bits.

For a transition on the trellis from time t to $t+1$ from state s' to s, BCJR computes the joint probability, given the observed output $\mathbf{o}$, as

$$P(S_t, S_{t+1}, \mathbf{o}) = P(S_t = s', S_{t+1} = s, \mathbf{o})\,. \tag{2.50}$$

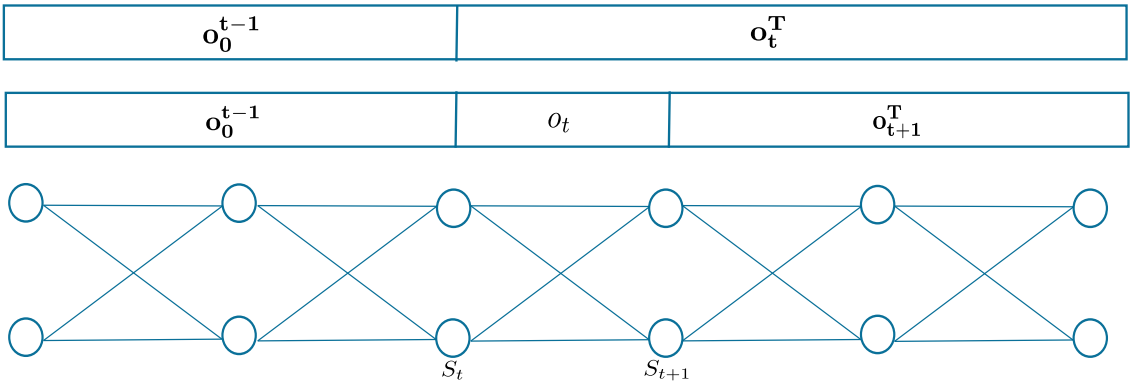


Figure 2.7: Trellis diagram for BCJR algorithm APP calculations

For every state at every step, it computes the joint probability given in Eq. (2.50) by computing three terms,

- Forward Recursion
- Backward recursion,
- Smoothed transition.

The forward recursion term computes the joint probability of the state S_t and the observations up to that time, $P(S_t, \mathbf{o}_0^{t-1})$. This computation is recursive, depending on the value of $P(S_{t-1}, \mathbf{o}_0^{t-2})$, this is derived in the following. Similarly, the backward recursion term is $P(\mathbf{o}_t^T|S_t)$ is computed depending on $P(\mathbf{o}_{t+1}^T|S_{t+1})$. The smoothed transition term relates the two states at times $t, t+1$.

We simplify Eq. (2.50),

$$\begin{aligned} P(\lambda_t, \mathbf{o}) &= P(S_t, S_{t+1}, \mathbf{o}) \\ &= P(S_t, \mathbf{o}_0^{t-1}) \cdot P(S_{t+1}, o_t|S_t, \mathbf{o}_0^{t-1}) \cdot P(\mathbf{o}_{t+1}^T|S_{t+1}, S_t, \mathbf{o}_0^t) \\ &= P(S_t, \mathbf{o}_0^{t-1}) \cdot P(S_{t+1}, \mathbf{o}_t|S_t) \cdot P(\mathbf{o}_{t+1}^T|S_{t+1}). \end{aligned} \tag{2.51}$$

The left and right factors of Eq. (2.51) are the forward and backward recursions, given below.

$$\begin{aligned} \underline{P(S_t, \mathbf{o}_0^{t-1})} &= \sum_{S_{t-1}} P(S_t, S_{t-1}, \mathbf{o}_0^{t-1}) \\ &= \sum_{S_{t-1}} P(S_{t-1}, \mathbf{o}_0^{t-2}) \cdot P(S_t, o_{t-1}|S_{t-1}, \mathbf{o}_0^{t-2}) \\ &= \sum_{S_{t-1}} \underline{P(S_{t-1}, \mathbf{o}_0^{t-2})} \cdot P(S_t, o_{t-1}|S_{t-1}) \ , \end{aligned} \tag{2.52}$$

and

$$\begin{aligned}
\underline{P(\mathbf{o}_t^T|S_t)} &= \sum_{S_{t+1}} P(\mathbf{o}_t^T, S_{t+1}|S_t) \\
&= \sum_{S_{t+1}} P(o_t, \mathbf{o}_{t+1}^T, S_{t+1}|S_t) \\
&= \sum_{S_{t+1}} P(\mathbf{o}_{t+1}^T|S_{t+1}, S_t, o_t) \cdot P(S_{t+1}, o_t|S_t) \\
&= \sum_{S_{t+1}} \underline{P(\mathbf{o}_{t+1}^T|S_{t+1})} \cdot P(S_{t+1}, o_t|S_t) \ . \qquad (2.53)
\end{aligned}$$

We now derive the smoothed transition term, which is used in both the forward and backward recursions,

$$P(S_t, o_{t-1}|S_{t-1}) = P(o_{t-1}|S_t, S_{t-1}) \cdot P(S_t|S_{t-1}) = P(o_{t-1}|S_t, S_{t+1}) \cdot P(S_t|S_{t-1}) \ . \quad (2.54)$$

The forward recursion in (2.52) is denoted by $\alpha_t(s')$, the backward recursion in (2.53) by $\beta_{t+1}(s)$, and the ones in (2.54), by $\gamma(s', s)$.

The forward and backward transitions have to be initialized in order to perform the recursive computations. The initialization depends on the system model. For convolutional codes, the starting state of the encoder is the all zero-state by convention, and the trellis is also terminated (by appending termination bits, for example) to the zero-state. In such cases,

$$\alpha_0(s = 0) = \beta_T(s = 0) = 1. \qquad (2.55)$$

A detailed derivation and comments on BCJR algorithm is given in [14].

Chapter 3

LDPC Decoding for Sources with Memory

In this chapter, the belief propagation decoding of LDPC codes is modified to include source memory. We consider a source which is modeled by a Markov chain and use the properties of the Markov model as additional information in the decoding. There are three decoding algorithm presented, two of which use direct 'left-to-right' links in the Tanner graph for forwarding the Markov dependencies in the source sequence in conjunction with a sum-product algorithm (SPA) decoder. The other method is a Turbo-like decoding scheme in which a BCJR decoder and an LDPC decoder iteratively exchange information to estimate the source sequence and use this information for decoding in SPA iterations, respectively. We show the simulation results for these three decoders for different transition probability matrices of the Markov model.

Combining a source with memory and a channel encoder leads to a description of a joint source-channel code. The chapter begins with an introduction to Shannon's source and channel coding theorems and current work in this field. We then describe the source model used, followed by the three decoding techniques and results. The chapter is concluded by outlining future steps.

3.1 Shannon's Source and Channel Coding Theorems

The first block in a transmission system is source compression or source coding, as shown in Fig. 2.1.

Source Coding: Source coding is the compression of source data to remove redundancy. There are two approaches to source coding,

- *lossy*: some information is lost via the process of compression and cannot be reconstructed.
- *lossless*: the original information can be fully reconstructed.

Various different approaches can be used for compression. **Entropy coding** is a loss-less method which assigns codes to source data symbols proportional to the entropy of the symbols. Since Shannon's information measure is negatively proportional to the probability of occurrence of a symbol, the resulting codes assign fewer bits to source data symbols which occur more frequently. This method results in fixed-to-variable length encoding. Huffman codes [38] and arithmetic codes are examples of entropy coding.

There are also dictionary-based methods, which are able to use the statistics and structure of the input source to create a dictionary which is built up dynamically such as the Lempel-Ziv type codes [39–41]. One of the ways to distinguish between different algorithmic approaches is by considering source modeling methods. Dynamic models are created/updated during runtime and fixed models, such as used in entropy coding require statistics of the data source to be known beforehand. Depending on the output of the algorithm used, source codes can also be categorized as fixed-to-variable length codes, variable-to-variable length codes, fixed-to-fixed, and variable-to-fixed length codes.

In [42], Shannon presented theorems on a discrete source communicating over a noisy channel and derived rate criterion for communications based on the entropy of the source and the capacity of the channel. Two 'theorems' have been defined for source and channel coding based on [42]. The theorem statements are based on assuming a joint source-channel code which has rate r.

Source-coding Theorem:

Let the entropy of a source U be $H(U)$. Reliable reconstruction of the source message is possible for $r > H(U)$.

Noisy Channel-coding Theorem:

Let the capacity of the transmission channel be C in bits/transmission or bits/time. Perfectly error-free transmission over a channel with capacity C is only possible if $r < C$.

Both the source-coding theorem and the noisy channel-coding theorem statements hold only for $N \to \infty$ and additionally, it can be proven from the source, and noisy-channel coding theorems that reliable transmission is possible by separating source and channel coding. The implication of [42] is that either reliable transmission is possible for a source U with entropy $H(U)$ over a channel with capacity C (for codeword length approaching infinity), or it is not possible at all, regardless of whether source and channel coding is performed jointly or separately. This decomposition leads to the principle known as the source-channel separation theorem.

Due to the separation theorem, source coding and channel coding are often carried out in tandem for many practical cases. However, the asymptotic assumption is not practical and the separation principle was shown to not hold for multi-user channels [43]. The transmissibility and consequently, the separation principle are also obtained under idealized assumptions of no delay and infinite computational power. When considering practical systems constrained by these issues, optimality for the separation theorem also does not hold. In this light, Joint Source-Channel (JSC) coding is an area of investigation.

Since the focus of this thesis is not directly JSC, in the following, we will use an uncompressed source sequence directly in the channel encoder-decoder. The goal of the thesis work is to include and observe the effects of dependencies in different parts of the communication system in the LDPC decoder. For applying this principle in the source part, the natural description leads to a JSC code, hence we provide a brief overview, in the following Section 3.2.

3.2 Joint Source-Channel Coding

The proofs in [42] were obtained by considering a discrete memoryless source (DMS) transmitting over a discrete memoryless channel. Additionally, the source is required to be ergodic and stationary. In [44], further results were obtained regarding reliable transmission bounds for general classes of sources and channels. The authors showed that the separation theorem does not hold for all sources whose entropy is less than the channel capacity, thus showing that the source-channel separation does not only fail in the finite block-length regime or multi-user scenarios. The authors further derive the requirements for the separation theorem for a wide variety of channels for any source, specifically, by relaxing the requirements of memorylessness

and stationarity of the source. A specific conclusion drawn is that for stationary sources, the separation-theorem holds for all channels. This result is of particular significance for this thesis since we consider a stationary stochastic source having memory.

One strategy for joint source-channel coding focuses on the dynamic allocation of bits between the source and channel encoder to minimize overall errors at the receiver. This strategy is useful for wireless fading channels, where the channel statistics are not time-invariant and thus require dynamic methods. In [45] the optimal bit allocation between source and channel codes are achieved by considering two additional degrees of freedom, the transmit power and data rate (constrained by an average power requirement). The authors propose an adaptive coded modulation scheme which minimizes end-to-end distortion and obtain an upper bound of the distortion for constant channel error rates. Authors in [46] propose machine learning for finding source and channel codes, given a computational budget and fixed block-length.

LDPC codes have been studied widely for JSCC systems. In [47], a compound Tanner graph structure for JSCC was proposed, and in [48], it was first used for JSCC. Named a D-LDPC (double-LDPC) code, the first code of the serial concatenated architecture performs syndrome source compression and the second one performs forward error correction. Message passing decoding on such a structure produces error floors. For mitigating the high error floors exhibited by the system, an improved connection profile between the graphs as well as shortening were employed, in [49] and [50], significantly improving performance.

Protograph-LDPC codes were suggested for JSCC in [51]. The performance of DP-LDPC (double Protograph-LDPC) for JSCC is analyzed for source statistics in [51–53].

Different Turbo decoding schemes for JSCC were analyzed in [54–56]. The authors of [54, 55] consider a hidden Markov model as source and consider uncompressed transmission over the channel. The turbo decoding scheme of [55] is comprised of parallel concatenated convolutional codes. The source statistics are used in the decoding by employing a modified trellis diagram, which jointly describes the source and channel code in one of the decoders. In order to reduce complexity, in [54], a separate decoder block for the source is considered which iterates information with the other blocks. A procedure for parameter estimation of the hidden Markov model is also shown. In [57], a similar Turbo structure for JSCC using LDPC codes is presented which also combines a hidden Markov model parameter estimation along with the decoder.

In [56], a serial concatenation structure is studied, which constitutes three codes, a Markov

chain of symbols, a variable length source encoder, and a convolutional channel code. The three corresponding decoder blocks interact in an iterative fashion.

Additionally, in [58], the general approach of channel coding for an uncompressed source is studied, the paper focuses on estimating parameters of the source at the decoder. In [59], an approach for using residual redundancy of the compressed source at the decoder is presented, modified Viterbi and soft-output Viterbi algorithms (SOVA) are proposed, which are able to exploit source symbol correlations at the decoder, for codes possessing a binary trellis representation.

3.3 Description of the Source Model

Our system model considers a source data stream generated by a discrete-time Markov chain.

Stochastic Markov models are useful to model and generate data sets which have various real world applications. Usually data generated by higher order models more accurately match real world data since more dependencies are captured. High order models can generate fragments of human speech, they can be used to model genes in DNA sequences, as well as user behavior on the World Wide Web, i.e., webpage sequences. In this context, we chose Markov models as generating the source data for our system. We start the following by deriving decoding steps in the decoder for an uncompressed Markov source of order-1 and subsequently move on to higher order models.

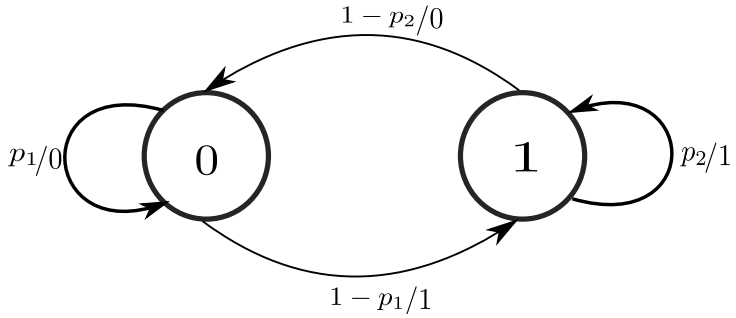


Figure 3.1: First-order Markov source

A Markov sequence can be described as generated by an auto-regressive state machine, meaning the output of the previous state becomes part of next state. For models of order-1, the output is directly the next state. The considered binary system of order-1 is given by a 2-state Markov model, shown in Fig. 3.1 and the corresponding transition probability matrix is

$$\mathbf{Q}_s = \begin{bmatrix} p_1 & 1-p_1 \\ 1-p_2 & p_2 \end{bmatrix} . \tag{3.1}$$

A DTMC described by Eq. (3.1) where $p_1 \neq p_2$ is an *asymmetric* model. The redundancy in such a model comes from both the memory and a non-uniform steady state distribution of source symbols. When, $p_1 = 1 - p_2$ for an asymmetric model, the memory component disappears and the only redundancy is in the unequal distribution of source symbols. For a symmetric model where $p_1 = p_2 \neq 1/2$, the redundancy in the source is only due to the memory. We consider such symmetric sources.

3.4 Incorporating the Source Model into Decoding

At the decoder, the source model can be used in conjunction with intermediate decoding results to compute 'extra' information which aids in decoding. In Fig. 3.2, the edges carrying such information are shown connecting the information variable nodes (in blue). The directed edges, from left to right, model the generation sequence of the source symbols. The derivation for the LLR on these edges are provided next.

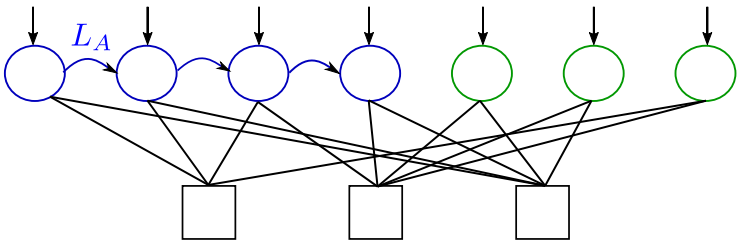


Figure 3.2: Modified Tanner graph with directed edges left-to-right between information nodes for a-priori information forwarding.

A source sequence $\mathbf{u}_{1:K}$ is generated by a model $\mathbf{Q}_s$, where the source bits $u \in \{0, 1\}$. The codeword $\mathbf{c}$ is generated by the systematic triangular parity-check matrix of the channel code. The transmit vector $\mathbf{x}$ is obtained by the anti-podal mapping given in Eq. (2.32). Transmission is done using BPSK modulation, with the signal power E_s normalized to 1.

An a-priori LLR value for x_q, the q^{th} bit of the length K information part of the codeword $\mathbf{x}$ is computed as $L_A(x_q) = \ln\frac{P(x_q=+1)}{P(x_q=-1)}$, according to Eq. (2.35). The node to the left of node q is indexed by $q-1$ with $q \neq 0$. This following derivation was given in [60].
Note, the probabilities given by $\mathbf{Q}_s$ represent transitions between source bits $u \in \{0, 1\}$, however, due to the one-to-one mapping according to Eq. (2.32), $p_1 = P(u_q = 0|u_{q-1} =$

$0) = P(x_q = +1|x_{q-1} = +1)$ and $p_2 = P(u_q = 1|u_{q-1} = 1) = P(x_q = -1|x_{q-1} = -1)$.

$$
\begin{aligned}
&L_A(x_q) \\
&= \ln\left[\frac{P(x_q = +1)}{P(x_q = -1)}\right] \\
&= \ln\left[\frac{P(x_q = +1|x_{q-1} = +1) \cdot P(x_{q-1} = +1) + P(x_q = +1|x_{q-1} = -1) \cdot P(x_{q-1} = -1)}{P(x_q = -1|x_{q-1} = +1) \cdot P(x_{q-1} = +1) + P(x_q = -1|x_{q-1} = -1) \cdot P(x_{q-1} = -1)}\right] \qquad (3.2) \\
&= \ln\left[\frac{P(x_q = +1|x_{q-1} = +1) \cdot \frac{P(x_{q-1}=+1)}{P(x_{q-1}=-1)} + P(x_q = +1|x_{q-1} = -1)}{P(x_q = -1|x_{q-1} = +1) \cdot \frac{P(x_{q-1}=+1)}{P(x_{q-1}=-1} + P(x_q = -1|x_{q-1} = -1)}\right] \\
&= \ln\left[\frac{p_1 \cdot \exp(L(x_{q-1})) + (1 - p_2)}{(1 - p_1) \cdot \exp(L(x_{q-1})) + p_2}\right] \qquad (3.3)
\end{aligned}
$$

Here, $L(x_{q-1})$ is the combined (decision) LLR at the variable node v_{q-1} representing transmitted bit x_{q-1}, as per Eq. (2.34). For this intermediate decision, all incoming edges to the variable node are added to obtain the decision at this iteration. The LLR given by Eq. (2.34) consists of three terms,

$$L = L_{intrinsic} + L_{a-priori} + L_{extrinsic} \, .$$

Extrinsic and intrinsic information are respectively associated with the parity-check constraints of the code and the channel properties, and are obtained by BP iterations on the Tanner graph of the code and the received analog values, respectively. The a-priori information for a symmetric Markov model can be computed by Eq. (3.3). The source Markov model parameters are assumed to be available at the decoder.

For decoding, at every iteration of the BP algorithm, a-priori information is required at all information nodes. For this, the current decision LLR at every information node is calculated and stored. For the next iteration, we use the (intermediate) decision LLR from the previous iteration to provide an a-priori value for each bit of the information sequence, from left to right, according to Eq. (3.3). BP decoding is then carried out according to Alg. 2. The parity bits of the codeword have no a-priori information.

The outgoing message from variable node q to check node w at iteration l for reflecting the

change in a-priori information at every iteration is

$$L^{(l)}_{(v_q \to w)} = L(y_q|x_q) + \sum_{w'} L^{(l-1)}_{(w' \to v_q)} + \ln\left(\frac{p_1 \cdot \exp(L(x_{q-1})^{(l-1)}) + (1-p_2)}{(1-p_1) \cdot \exp(L(x_{q-1})^{(l-1)}) + p_2}\right) , \tag{3.4}$$

where y_q is the received analog value at variable node q. The estimate of the left node does not necessarily need to be from the previous iteration. We want an updated estimate of the left node, we could also take it from the decision LLR at the current iteration. This is an implementation choice, the results compiled here are for estimates from the previous iteration. Furthermore, blocked transmission is assumed, one codeword is transmitted at a time. Since there is no node to the left for the first bit of every codeword, the a-priori LLR is taken as the logarithmic ratio of the steady state probabilities. For the first iteration, when only intrinsic information is available, all nodes are initiated with a-priori LLR, $L_{A_0}(x_q)$, by using the ratio of the steady state probabilities (instead of the right term in Eq. (3.4))

$$L_{A_0}(x_q) = \ln\left(\frac{P(u=0)}{P(u=1)}\right) = \ln\left(\frac{P(x=+1)}{P(x=-1)}\right) . \tag{3.5}$$

For computing the modified a-priori LLRs, in the numerator of Eq. (3.3), a summation is performed for all source model state transitions which produce an output $u = 0$ $(x_q = +1)$. Similarly, for the denominator, the sum is over all state transitions which produce an output $u = 1$ $(x_q = -1)$. For an order-1 model, only the previous output in the source sequence influences the current output. We define the 'memory' of a discrete time Markov chain to be m. For memory m, for a binary alphabet, there are 2^m sequences of length m which determines the output, which are also the states. The transition probability matrix has dimension $2^m \times 2^m$. Thus for finding a generalized formula for the modified a-priori information for arbitrary m, in the numerator (denominator), we sum over all the state transitions which produce an output $u = 0$ $(u = 1)$ for all 2^m states.

$$L_A(u_q) = \ln \frac{\sum_{s' \in 2^m} P_{s' \to s}(u=0) P(s')}{\sum_{s' \in 2^m} P_{s' \to s}(u=1) P(s')} , \tag{3.6}$$

where, $P_{s' \to s}(u = 0)$ are all state transitions from s' to s which produce an output $u = 0$, and analogous for $P_{s' \to s}(u = 1)$. Thus, the decoding easily generalizes to arbitrary order-m Markov models. For the simulation results provided in Section 3.5, we restrict ourselves to order-1 and order-2 Markov chains.

3.4.1 Simplified Computation for Source Model Inclusion in Decoding using Jensen's Inequality

The computation provided in Eq. (3.2) can be simplified using Jensen's inequality. Jensen's inequality states, if $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$ and B is a random variable, then for a concave function ϕ

$$\sum_i \lambda_i \cdot \phi(\mathrm{b}_i) \leq \phi(\sum_i \lambda_i \mathrm{b}_i) \; . \tag{3.7}$$

Re-writing Eq. (3.2) yields

$$\begin{aligned} L_A(x_q) = & \underbrace{\ln\left[P(x_{q-1}=+1)\cdot p_1 + P(x_{q-1}=-1)\cdot(1-p_2)\right]}_{a} \\ & \underbrace{-\ln\left[P(x_{q-1}=+1)\cdot(1-p_1) + P(x_{q-1}=-1)\cdot p_2\right]}_{b} \end{aligned} \tag{3.8}$$

We now apply Eq. (3.7) to a and b from Eq. (3.8) and respectively obtain

$$\underbrace{P(x_{q-1}=+1)\cdot\ln(p_1) + P(x_{q-1}=-1)\cdot\ln(1-p_2)}_{c}$$

and

$$\underbrace{-[P(x_{q-1}=+1)\cdot\ln(1-p_1) + P(x_{q-1}=-1)\cdot\ln(p_2)]}_{d} \; .$$

Thus, $L_A(x_q)$ can be approximated by

$$\begin{aligned} L_{A,\text{Jensen}}(x_q) = & \underbrace{P(x_{q-1}=+1)\cdot\ln(p_1) + P(x_{q-1}=-1)\cdot\ln(1-p_2)}_{c} \\ & \underbrace{-P(x_{q-1}=+1)\cdot\ln(1-p_1) - P(x_{q-1}=-1)\cdot\ln(p_2)}_{d} \; . \end{aligned} \tag{3.9}$$

The required $P(x_{q-1}=\pm 1)$ from Eq. (3.9) are obtained by

$$P(x_{q-1}=+1) = \frac{\exp L(x_{q-1})}{1+\exp L(x_{q-1})} \; , \quad \text{and} \tag{3.10}$$

$$P(x_{q-1}=-1) = \frac{1}{1+\exp L(x_{q-1})} = 1 - P(x_{q-1}=+1) \; ; \tag{3.11}$$

where, at the $(q-1)^{th}$ variable node v_{q-1}, the decision LLR for x_{q-1} is given by $L(x_{q-1})$. According to Eq. (3.7), c and d from Eq. (3.9) are lower and upper bounds of a and b from Eq. (3.8), respectively[1]. Thus, Eq. (3.9) is an approximation of Eq. (3.8). We rewrite Eq. (3.9)

[1] The definitions of b and d include the minus signs.

by gathering terms

$$L_{A,\text{Jensen}}(x_q) = P(x_{q-1} = +1) \cdot \ln\left[\frac{p_1}{1-p_1}\right] + P(x_{q-1} = -1) \cdot \ln\left[\frac{1-p_2}{p_2}\right] . \quad (3.12)$$

Computing a-priori values using Eq. (3.9) is computationally simpler, as $\ln[\frac{p_1}{1-p_1}]$ and $\ln[\frac{1-p_2}{p_2}]$ are fixed values. Hence, the expression is linear in $P(x_{q-1} = +1)$ or $P(x_{q-1} = -1)$. As shown in the simulation results, both expressions for a-priori information provide similar performance.

3.4.2 Turbo-like Decoding Scheme

[2]A hidden Markov model, described in Section 2.1.2 is characterized by a state transition matrix and an emission matrix. In HMM literature, a classic problem is to determine the state sequence, given the observed output sequence and the HMM model, Θ. Both Viterbi and BCJR decoders can be used for this task. In this section, we reformulate our system as a HMM.

On the decoder side, the received analog values are the output of the Markov state machine corrupted by additive i.i.d. Gaussian noise with zero mean and standard deviation σ_n. We consider this observed sequence to be the continuous output of an HMM model. The received values are viewed as samples from a continuous emission probability density, given by a Gaussian density $\mathcal{N}(\pm 1, \sigma_n^2)$ for states $x \in \{\pm 1\}$. Since the decoder only observes the analog outputs, and has knowledge of the channel statistics as well as $\mathbf{Q}_s$, it can be viewed as a classical problem of finding the sequence of states of the system. However, it is not necessary to view this as a HMM problem. We know the information vector $\mathbf{u}$ is generated by a state-machine. Trellis based decoding methods such as Viterbi (Alg. 1) or BCJR (given in Section 2.2.5.1) can both be utilized for finding the sequence of states, given that the transition probabilities between states ($\mathbf{Q}_s$) are known and outputs are given.

To this end, we construct a decoder architecture similar to a serial concatenated Turbo decoding scheme. Our goal is to use the BCJR algorithm to deduce the probable state sequence and use this a-priori information in the LDPC decoder. The LDPC code performs parity-checks and provides information for the BCJR decoder. The system is described in Fig. 3.3.

The outer Markov state machine block produces a length K information sequence. The LDPC encoder generates codewords $\mathbf{x}$ of length N. After transmission, on the decoder side, the channel intrinsic information of the received vector $\mathbf{y}$ is fed into the LDPC decoder, along with the a-priori estimate, computed by the BCJR decoder. One iteration of the decoder

[2]Portions of Sections 3.4.2 and 3.5 are taken directly from: N. S. Islam and W. Henkel, "Information Forwarding in LDPC Decoding for Markov Sources," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, 2018, pp. 1–5

is counted as serial decoding performed by the LDPC decoder first and subsequently, by the BCJR decoder. In the first iteration, there is no incoming information from the BCJR decoder, hence $L_A(x_q) = 0$. The output of the LDPC decoder provides a-priori information for the BCJR decoder, after subtracting the input the BCJR decoder provided in the previous iteration, to avoid information recycling, and vice-versa for the LDPC decoder.

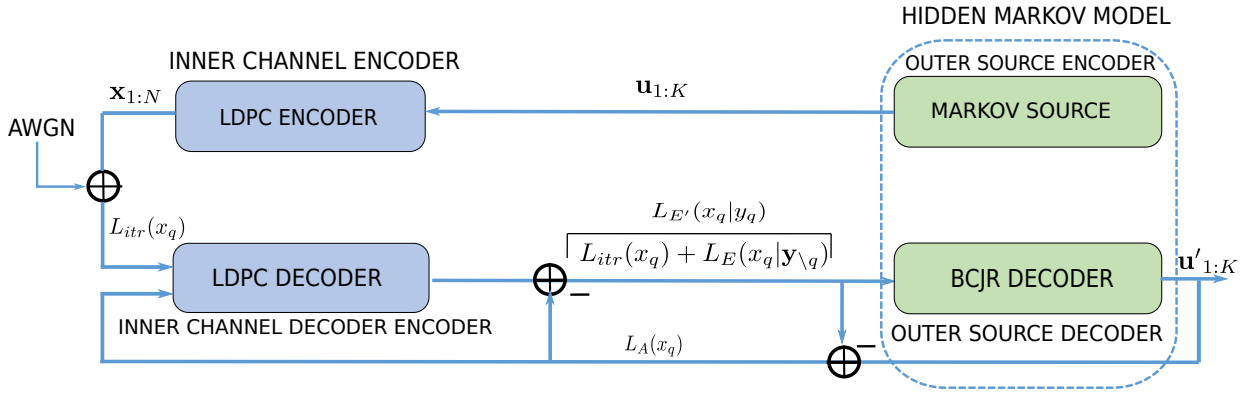


Figure 3.3: Concatenated encoder and decoder structure

We start our iterations at the LDPC decoder, with no a-priori information, yet. The Turbo-scheme is a standard serially concatenated Turbo coding scheme. The BCJR decoder is not directly connected to the channel and does not have access to the intrinsic information. The values for $\alpha_1(s)$ and $\beta_K(s)$ are initialized to the steady-state probabilities. The BCJR computes a MAP LLR value for every segment of the trellis.

The information exchange within the serially concatenated code is shown in Fig. 3.4. For an order-1 model, the trellis has 2-states and the outputs of the Markov model become the states for the next time step. As described, iterations start in the inner LDPC decoder which estimates the values of the information sequence at the variable nodes. These estimates map onto the state transitions in the trellis. Solid lines represent an output $+1$ and dashed lines represent an output -1. The probabilities of being $+1$ and -1 are computed from the outgoing $L_{E'}(x_q|y_q)$ of the LDPC decoder and placed on the corresponding trellis paths as shown by the arrow from the variable nodes to the state transitions. This extrinsic information computed by the LDPC decoder, $L_{E'}(x_q|y_q)$, is obtained after subtracting the information the BCJR decoder provided, it is extrinsic information to the BCJR decoder itself.

3.5 Performance Comparison

In this section we provide simulation results. Preliminary results were published by us in [61]. An irregular rate-$1/2$ LDPC code was constructed using the following variable- and check-node

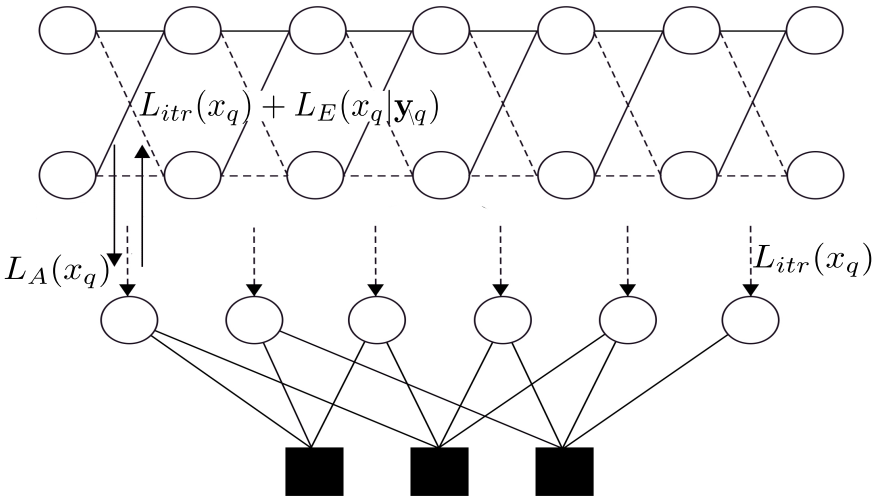


Figure 3.4: Information exchange between the LDPC and BCJR decoders

degree distribution polynomials,

$$\begin{aligned} \lambda(x) &= 0.28286x + 0.39943x^2 + 0.31771x^7 \ , \\ \rho(x) &= 0.6x^5 + 0.4x^6. \end{aligned} \tag{3.13}$$

The PEG algorithm was used to construct a systematic triangular parity-check matrix, $\mathbf{H}$. BER results were compiled after 100 independent erroneous words were found. The BER curves are presented in figures 3.5 and 3.6, for maximum number of iterations 10 and 20, respectively. For ease of reference, the decoding algorithms presented in Section 3.4 and sub-sections 3.4.1, and 3.4.2 will be referred to as, Dec-1, Dec-2, and Dec-Ser. As reference, performance curves of an LDPC decoder using the sum-product algorithm are provided (in which the a-priori information according to the source Markov model is not incorporated), shown in blue and green, labeled Ref $\mathbf{Q}_{s1}$ and Ref $\mathbf{Q}_{s2}$. These curves are generated for matrices

$$\mathbf{Q}_{s1} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix} \ , \ \mathbf{Q}_{s2} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \ .$$

The transition probabilities $p_1 = p_2 \triangleq p$ are listed in the legend of the plots. As expected, since SPA decoding is constructed for sources without redundancy, the performance curves for $p = 0.5$ and $p = 0.9$ are identical.

The results for Dec-1 are provided for transition matrices $\mathbf{Q}_{s1}$, $\mathbf{Q}_{s3}$ and $\mathbf{Q}_{s4}$, where

$$\mathbf{Q}_{s3} = \begin{bmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{bmatrix} \ , \ \mathbf{Q}_{s4} = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix} \ .$$

We observe from the results that incorporating memory into decoding improves the performance significantly. The lower the entropy of the source model, the more benefit there is in including the a-priori estimates, since the memory between neighboring bits is stronger. Results for Dec-2 are plotted for $\mathbf{Q}_{s1}$. We observe that the performance curves computed by the two methods are identical. Results for the order-2 (O:2) Markov model are also provided, for $\mathbf{Q}_{s5}$, labelled $P_{\{0,0\}}$, using Dec-1.

$$\mathbf{Q_{s5}} = \left[\begin{array}{c|cccc} & (0\mathbf{0}) & (1\mathbf{0}) & (0\mathbf{1}) & (1\mathbf{1}) \\ \hline (00)| & 0.9 & 0 & 0.1 & 0 \\ (10)| & 0.5 & 0 & 0.5 & 0 \\ (01)| & 0 & 0.5 & 0 & 0.5 \\ (11)| & 0 & 0.1 & 0 & 0.9 \end{array}\right]$$

The states of $\mathbf{Q}_{s5}$ are also output sequences. 00 refers to the output sequence $[0,0] = [u_{q-2}, u_{q-1}] = [x_{q-2}, x_{q-1}] = [+1,+1]$. The state transitions are listed in the format row-to-column, meaning, the $(1,1)$ element of $\mathbf{Q}_{s5}$ refers to the state transition from $\{0,0\}$ to $\{0,0\}$, the bit marked in red being the output, i.e., the next information bit.

For the Turbo decoding scheme, serial decoding was performed 10 times, by iterating between the LDPC and BCJR decoders, each only performing one iteration internally. The iterations are started at the LDPC decoder. After this iterative round, the resultant estimate of the a-priori LLR from the BCJR decoder, $L_A(x_q)$ was then used in a regular LDPC decoder; which does not consider the memory of the sequence; for a maximum of 10 iterations. The total number of iterations inside the Tanner graph in the Turbo scheme is then 20. In order to have a fair comparison, Dec-1 was called with 20 iterations for the same Markov model $\mathbf{Q}_{s1}$.

We observe that the serial architecture performs better at low SNRs. However, the curve is not as steep as the BER curves for Dec-1. In the Turbo-decoding scheme, scheduling is an important issue. From the simulations, it was observed that iterating between the two constituent decoders, each only performing one internal iteration, does not provide benefits to the Turbo scheme. The BP decoder needs to run some iterations internally after the BCJR decoder.

We conclude that when the correlation between the source bits is high, including the source

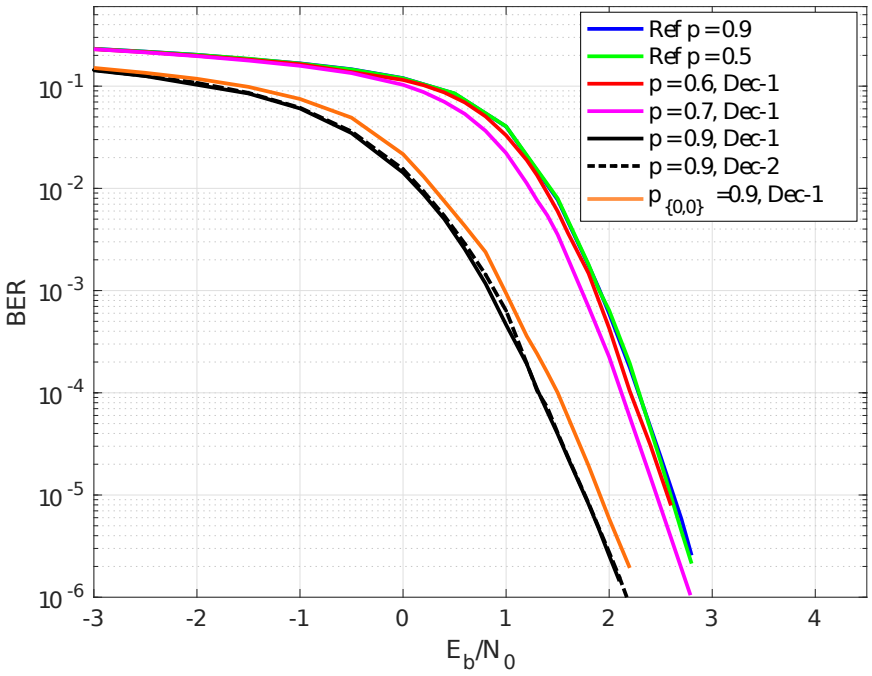


Figure 3.5: Simulation results for Dec-1, Dec-2 for different $\mathbf{Q}_s$ matrices for a maximum of 10 iterations

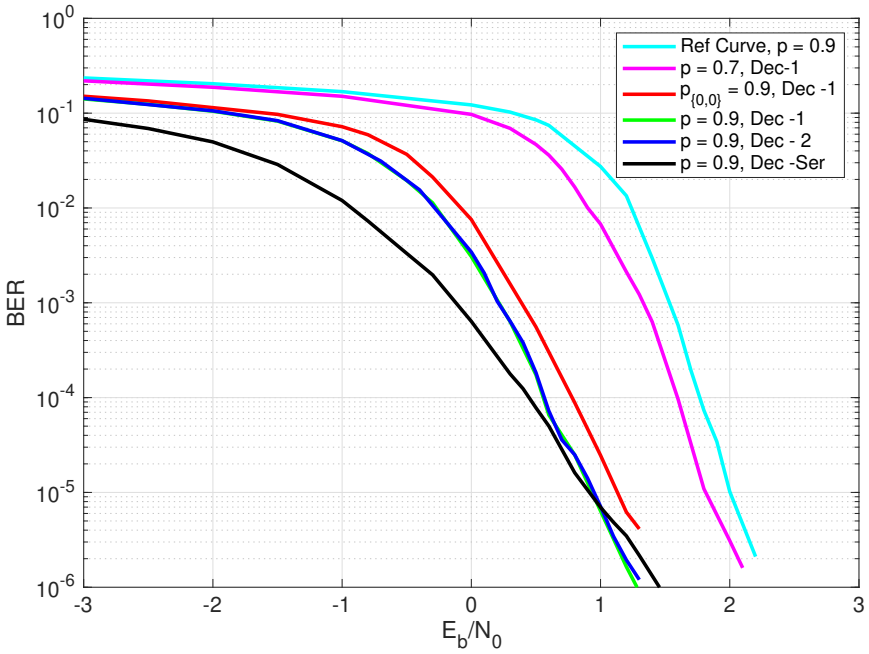


Figure 3.6: Simulation results for Dec-1, Dec-2, and Dec-Ser for different $\mathbf{Q}_s$ matrices for a maximum of 20 iterations

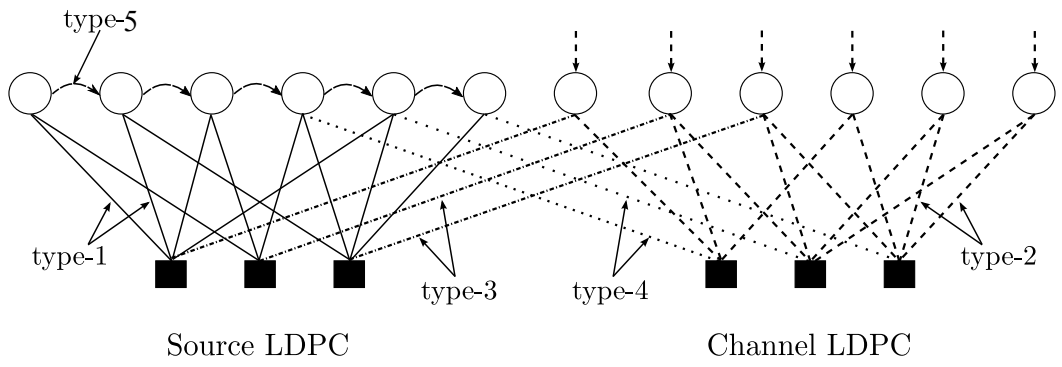


Figure 3.7: JSC scheme with Markov links ©2018 IEEE

dependency leads to substantial gains in decoding. Since the complexity of the *inside-Tanner-graph* decoding algorithms for including source redundancy is not high, these methods are useful, specifically the method from Section 3.4.1. We also observe that the Turbo-scheme provides useful gains at low SNR regimes, where the Tanner graph based methods do not yet start their convergence.

3.6 Summary and Future Steps

Modifying message passing decoding to include source memory improves the performance of LDPC codes. The source model can be incorporated to provide a-priori information directly inside the Tanner graph or by using a concatenated scheme. The gains obtained by using source memory are dependent on the entropy of the source model, lower entropy sources providing more gains.

We have considered here a direct transmission of an uncompressed source via an error correcting code. Such a scheme is inefficient when considering overall data throughput. The D-LDPC codes discussed at the beginning of this chapter address this shortcoming by using syndrome source compression first and then perform error correction coding. While the variable nodes or source information for such a graph is not transmitted, the goal is to reconstruct the source symbols by erasure filling. However, as shown in Fig. 3.7, the left-to-right links, denoted edge-5, can be used to aid in decoding and is considered to be the next step, leading to a novel JSCC scheme.

Recently, during a discussion with Prof. Volker Kühn, he suggested to incorporate bidirectional links between the information nodes in the Tanner graph, since there is information to be gained from both directions regardless of the source generation direction. This would lead to a BCJR-like forward-backward estimation inside the Tanner graph. We gratefully acknowledge

his input and would also incorporate this step in the future. Another possible extension lies in expanding the memory model. For example, natural languages can be modeled by DTMCs. The non-binary message alphabet would lead to non-binary codes since the memory would be between symbols instead of bits.

Chapter 4

Optimization of LDPC Codes for Sources with Memory

We now focus on code optimization for the source model from Chapter 3. A decoder structure from the previous chapter consists of a modification to the Tanner graph such that the information variable nodes have an incoming edge from neighboring variable nodes, reflecting the source sequence modeled by a Markov chain. For optimizing a code having such a structure, the density evolution procedure has to be updated.

The chapter begins with discussing decoder and channel symmetry conditions which allows for assuming the transmission of the all-ones codeword. Under this assumption, the LLR message densities should move towards $+\infty$ as iterations in the decoder are increased. Instead of densities, the mutual information between the input to the channel and LLR messages in the decoder can also be tracked. For converging codes the mutual information should increase after every full iteration in the decoder. Under this convergence constraint, the rate $1/2$ code with lowest threshold is found. The required proportion distribution constraints and stability conditions are also presented. After the optimal code polynomials are found, the $\mathbf{H}$-matrix design is discussed for this structure. Then simulation results are provided showing the performance of the designed codes against standard codes.

4.1 Optimizing a Channel Code for a Source with Memory

In Chapter 3, three types of JSCC structures were mentioned, LDPC-code based JSCC, Turbo-coding based schemes, and methods based on dynamic allocation of source and channel code rates for fading channels. The compound Tanner graph structure for JSCC was optimized in both [49, 62]. In [49], the code structure is optimized in an iterative fashion, between channel code and source code, whereas, authors in [62] optimize the source and channel code jointly. In [63], the authors optimized a Turbo-coding structure for JSCC application, by making use of source statistics. The dynamic allocation algorithms adjust their rates in-situ.

The above methods do not correspond to the in-Tanner graph structure analyzed in Chapter 3 and therefore, we present an optimization method in here.

4.1.1 Code Design for Irregular LDPC Codes for Sources with Memory

The density evolution procedure for determining the performance of a code (λ, ρ) was briefly presented in Section, 2.2.3. In this section we will define the required concepts from the procedure and build codes suitable for our application using density evolution analysis.

In Section 2.2.3, it was mentioned that density evolution is based on some symmetry conditions from [13]. We further elaborate here. The evolution of LLR densities during decoding are influenced by the decoding algorithm (the operations at variable and check nodes) and the channel (influences the intrinsic LLR message). The symmetry conditions thus relate to the decoding algorithm and the channel.

The modified decoding algorithm from Section 3.4 adds an additional directed edge at the information variable nodes, as shown in Fig. 3.2. The operation at the variable nodes remains unchanged, an addition of incoming LLRs are performed. For the following analysis, we define

- Message maps (processing performed) at variable, and check nodes, at the l^{th} iteration, $\Psi_v^{(l)}$ and $\Psi_c^{(l)}$, respectively. The operations are still equations (2.38) and (2.39), respectively.

Variable-node Symmetry

The *variable-node symmetry* condition requires *sign-inversion in-variance*. If the sign of all incoming LLRs at a variable node are changed, the outgoing LLR sign also changes. For a node with degree d_v at the l^{th} iteration, incoming LLRs on edges are $L_{i\in\{1:d_v-1\}}$. We have,

$$\Psi_v^{(l)}(-L_0,-L_1,-L_2,-L_{d_v-1},...,-L_A) = -\Psi_v^{(l)}(L_0,L_1,L_2,L_{d_v-1},...,L_A)\ , \tag{4.1}$$

where L_0 and L_A refer to the intrinsic and a-priori information, respectively. The intrinsic information incoming at the variable node also has the same property, and is relevant for the first iteration, $\Psi_v^{(0)}(-L_0) = -\Psi_v^{(0)}(L_0)$.

Check-node and Channel Symmetry

The check node update remains unchanged hence the check node symmetry holds, from [13]. The symmetry condition for the channel requires output-symmetry, since the channel is a BI-AWGN channel, $p(y_t|x_t=+1) = p(-y_t|x_t=-1)$. Hence, channel output symmetry also holds.

Under the symmetry conditions, the error probability is independent of the actual transmitted codeword - which is a key insight and allows for assuming the transmission of any codeword to analyze the error correction characteristics of the code. This assumption simplifies the analysis of message evolution. Since it can be assumed without loss of generality that the all-ones codeword is transmitted; following the definition of LLRs; a converging code must have densities which move towards $+\infty$.

For analysis under density evolution, a key restriction for message passing decoders is that there *must* be an exclusion of the incoming message on an edge, when computing the outgoing message on the same edge. This constraint is related to the extrinsic information evolution for iterative decoders and allows for code analysis. The uni-directional additional direct-VN-linking edges do not alter this constraint at the decoder.

A design criterion to consider is that the parity portion of the codeword does not have a-priori information. This is handled when optimizing the code by allowing two classes of variable nodes (multi-edge) and will be described in the optimization constraints.

Thus, we can use density evolution as a tool to analyze the performance of our code. There are two main approaches to density evolution:

- For a desired fixed rate, find the lowest threshold code.
- For an SNR threshold, find the code which has the best rate.

In here, we have chosen the first criterion, fixing the code rate to be $1/2$ and finding the code with the best (lowest convergence) threshold.

4.2 Code Design Constraints

For finding the best code, different algorithms can be used to search for the optimum code under the constraints mentioned in this section. The density of LLR messages evolves during the iterative decoding process, which represents changes in the mutual information between the channel input and the decoder output at variable nodes. For successful decoding, the mutual information should increase after every iteration. We discuss this convergence constraint as well as other requirements in the following.

Proportion distribution constraints:

In our system, there are two kinds of variable nodes on the information side, the K systematic information nodes have a *left-to-right* a-priori information link between them as shown in Fig. 3.2. There is no a-priori information available for the parity nodes. Since there are two kinds of variable nodes, and the outgoing messages on edges from each class of nodes are different - this is classified as a multi-edge-type LDPC code with two classes of variable nodes, denoted by $j = 1$ for information nodes and $j = 2$ for parity nodes. The proportion distribution constraints governing the relationship between the degree polynomials λ and ρ are

$$\sum_{j=1}^{2} \sum_{i=2}^{d_{\mathsf{vmax}_j}} \lambda_i^{(j)} = 1 \,, \tag{4.2}$$

$$\sum_{i=2}^{d_{\mathsf{vmax}_2}} \frac{\lambda_i^{(2)}}{i} = \sum_{i=2}^{d_{\mathsf{cmax}}} \frac{\rho_i}{i} \,, \tag{4.3}$$

where d_{vmax_j} and d_{cmax} are the maximum degrees of variable and check nodes, respectively. The following equation relates $\frac{M}{K}$; a parameter related to the rate; and degree distributions

$$\sum_{i\geq 2}^{d_{\mathsf{vmax}_2}} \frac{\lambda_i^{(2)}}{i} = \frac{M}{K} \sum_{i\geq 2}^{d_{\mathsf{vmax}_1}} \frac{\lambda_i^{(1)}}{i} \,. \tag{4.4}$$

Stability Constraint:

The stability constraint derived in [26] for binary input AWGN channels is

$$e^{-\frac{1}{2{\sigma_n}^2}} < \frac{1}{\lambda'(0)\rho'(1)} \ , \tag{4.5}$$

where $\lambda^{'}(x)$ and $\rho^{'}(x)$ are the derivatives of the degree polynomials and ${\sigma_n}^2$ is the noise power (variance). $\lambda(x)$ is the joint degree polynomial for both classes. The stability constraint ensures that the error probability converges to zero for iterations approaching infinity for binary input symmetric output memoryless channels. Cycles in the code-graph having degree-2 nodes have a significant effect on the performance of a code, thus the constraint relates to the maximum number of degree-2 variable nodes.

Convergence Constraint:

For successful decoding and error probabilities approaching zero for the number of iterations going to infinity, the mutual information between the transmitted symbols and the corresponding LLRs is required to be increasing after every iteration in the decoder. To compute the average mutual information evolution of a code, first it is calculated for each outgoing edge of each variable node and then averaged over the entire graph.

This constraint is implemented in the optimization program by assuming an average outgoing mutual information at an iteration $(l-1)$ from the variable nodes, denoted by $x_{vc}^{(l-1)}$. The average mutual information for the next iteration $x_{vc}^{(l)}$ is then computed. A range of values are chosen for $x_{vc}^{(l-1)}$. This mimics the evolution along the EXIT chart curves for LDPC codes. The mutual information approaching 1 signifies error probability approaching zero. The convergence constraint requires $x_{vc}^{(l)} > x_{vc}^{(l-1)}$. The optimum $(\lambda(x), \rho(x))$ is then found that would adhere to all the code constraints.

In order to compute the mutual information $x_{vc}^{(l)}$, the check-to-variable node mutual information $x_{cv}^{(l-1)}$ is first computed using Eq. (2.49). Now, the combination at variable nodes is a summation of incoming LLR values representing the a-priori, intrinsic, and extrinsic information. For binary input AWGN channels, the symmetry condition for densities (Section 2.2.3) [27] holds and hence, the densities of the intrinsic LLR are consistent. It is typically assumed that the check node combinations preserve the consistency of the outgoing densities. For the intrinsic and extrinsic information message densities, it is then sufficient to track the means.

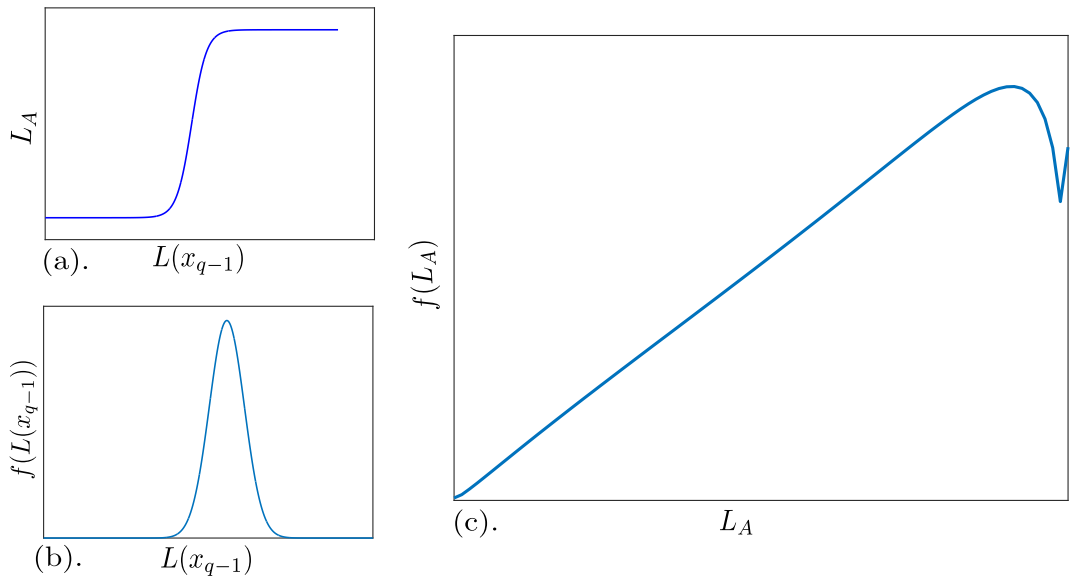


Figure 4.1: (a) Plot of L_A as a function of $(L(x_{q-1}))$ (b) Consistent Gaussian density of $L(x_{q-1})$ (c) Density of $L(A)$

We now consider the a-priori densities at the variables nodes. We recall the modified a-priori LLR from Eq. (3.3). The a-priori information is derived from the current estimate of the left node of information variable nodes, weighted by parameters of the transition matrix $\mathbf{Q}_s$. The weighting of the estimate $L(x_{(q-1)})$ translates into a density transform on the density of $L(x_{(q-1)})$, which no longer preserves the consistency properties of $f\left(L(x_{(q-1)})\right)$. Summation of LLR values means convolution of their respective densities. At the variable nodes, we perform actual convolutions of incoming LLR densities, i.e., the a-priori and the consistent intrinsic and extrinsic information densities.

The mentioned densities are sketched in Fig. 4.1 for $p = 0.7$. In Fig. (4.1)(b), the consistent density of the decision LLR at the variable nodes is shown. The decision LLR at a variable node is the summation of all edges incoming from check nodes and the intrinsic information. This is computed by the argument of the function g_t of Eq. (4.6) for a node of degree i. In order to find the density of a-priori values, this density is modified through the function given by Eq. (3.3), shown in Fig. 4.1 (a). The resultant density of the a-priori LLR is shown in Fig. (4.1)(c). The convolutions at the variable nodes to compute the outgoing density is then between a density (of the shape) shown in Fig. (4.1)(c) and a density given by the left term of Eq. (4.6), having the same shape as Fig. (4.1)(b).

The degree distribution polynomial of the information nodes is $\lambda(x)^{(j=1)}$ and for parity-nodes

(of variable nodes) it is $\lambda(x)^{(j=2)}$. At the variable nodes, the outgoing density on an edge of an information-node with degree i is

$$f\left(\frac{2}{\sigma_n^2}+(i-1)J^{-1}(x_{cv}^{(l-1)})\right) * \underbrace{g_t\left(\frac{2}{\sigma_n^2}+(i_1)J^{-1}(x_{cv}^{(l-1)})\right)}_{f_{i_1}(L_A):\forall\, i_1\in\{2:d_{\mathsf{vmax}_1}\}}, \tag{4.6}$$

where $f(\cdot)$ denotes a density. $f_{i_1}(L_A)$ is the a-priori density outgoing from a node with degree i_1. $g_t\left(\frac{2}{\sigma_n^2}+(i_1)J^{-1}(x_{cv}^{(l-1)})\right)$ refers to the density transform according to Eq. (3.3) on $f\left(L(x_{(q-1)})\right)$ (shown in Fig. 4.1(c)). The mutual information evolution per edge of variable node i is

$$\sum_{i_1\geq 2}^{d_{\mathsf{vmax}_1}} \lambda_{i_1}\cdot g_{mi}\left[\underbrace{f\left(\frac{2}{\sigma_n^2}+(i-1)J^{-1}(x_{cv}^{(l-1)})\right) * g_t\left(\frac{2}{\sigma_n^2}+(i_1)J^{-1}(x_{cv}^{(l-1)})\right)}_{f(v)}\right], \tag{4.7}$$

where $g_{mi}(\cdot)$, given by Eq. (4.9), is a function which computes the mutual information. Averaged over all information variable nodes of the graph, the mutual information is

$$\sum_{i\geq 2}^{d_{\mathsf{vmax}_1}} \lambda_i \sum_{i_1\geq 2}^{d_{\mathsf{vmax}_1}} \lambda_{i_1}\cdot \underbrace{g_{mi}\left[f\left(\frac{2}{\sigma_n^2}+(i-1)J^{-1}(x_{cv}^{(l-1)})\right) * g_t\left(\frac{2}{\sigma_n^2}+(i_1)J^{-1}(x_{cv}^{(l-1)})\right)\right]}_{x_{vc}}. \tag{4.8}$$

The information-node polynomial is the same, $\lambda_i=\lambda_{i_1}$ for $\forall i\in\{2:d_{\mathsf{vmax}_1}\}$ where d_{vmax_1} is the maximum number of degrees allowed for information nodes.

For the computation of $g_t(\cdot)$, the a-priori information is ignored in computing $L(x_{q-1})$. Including this inside the optimization would lead to a recursive function. To compute the mutual information x_{vc} from Eq. (4.8) between the discrete distribution of the transmitted binary message $P_X(x)$ and the density at output of the variable nodes $f(v)$, shown in Eq. (4.7), we use

$$\begin{aligned} x_{vc} &= g_{mi}(f(v)) \\ &= H(V)-H(V|X) \qquad (4.9)\\ &= -\sum_V\left[\left(\sum_{x=\pm1}P_X(x)f_{V|X}(v|x)\right)\log_2\left(\sum_{x=\pm1}P_X(x)f_{V|X}(v|x)\right)\right] \\ &\quad +\sum_V\sum_{x=\pm1}\left(P_X(x)f_{V|X}(v|x)\right)\log_2\left(f_{V|X}(v|x)\right) \qquad (4.10)\end{aligned}$$

The parity class (parity part of the variable nodes) of this multi-edge-type code follows the simplifications proposed in [27], given in Eq. (2.48). The optimization problem has a quadratic convergence constraint for the information variable nodes, and linear proportion distribution and stability constraints as well as a linear convergence constraint for parity nodes.

We start our optimization process by fixing $\rho(x)$ and finding the corresponding $\lambda(x)$ which adheres to rate-$1/2$ and finds the best threshold code. The optimization algorithm is provided in Alg. 4.
For the binary input AWGN channel, the relevant channel quality parameter is the signal to noise ratio (SNR) which is defined as $\frac{E_s}{N_0} = 10\log_{10}\frac{1}{{\sigma_n}^2}$, where N_0 is the two-sided noise power spectral density and E_s is the signal energy normalized to 1.

4.3 Constructing the H matrix

To construct the parity-check matrix $\mathbf{H}$, the zigzag-PEG algorithm from Section 2.2.4 is used.

Typically in the construction, length-4 cycles are the shortest possible cycle. In our modified Tanner graph from Fig. 3.2, due to the *left-to-right* link between information nodes, if neighboring variable nodes would be connected to the same check-node, a closed path of length-3 connecting the two variable nodes would result.

Although this link does not convey the same quality of information as an edge between variable and check nodes due to the attenuation of the LLR value as per Eq. (3.3), we construct a parity-check matrix avoiding such links and compare the performance to a matrix when this restriction is not in place. This restriction only applies to the information-variable nodes, the parity part of the codeword is not affected and the zigzag-construction at the parity side enabling easier encoding is still possible.

In Section 2.2.4, it was mentioned that when constructing a check-matrix for both (λ, ρ), constraints prohibiting connections to check-nodes which violate $\rho(x)$ have to be included. Due to the desired lower-triangular structure of the parity-check matrix, connections to check-nodes in the upper triangular part of the graph are also prohibited. Additionally, in order to prevent neighboring variable nodes from being connected to the same check-node, adjacent 1s in the same row of $\mathbf{H}$ are to be avoided. PEG establishes variable node connections from the last column of $\mathbf{H}$ or lowest degree VNs. While constructing the matrix, when constructing column v_i, v_{i+1} has been fully connected. Thus when placing edges for the VN v_i, we exclude

Algorithm 4: Finding Optimized Codes

for values of $x_{vc}^{(l-1)}$ **do**

$$x_{cv}^{(l-1)} = 1 - \sum_{j=2}^{d_{\mathsf{cmax}}} \rho_j J\left((j-1)\, J^{-1}\left(1 - x_{vc}^{(l-1)}\right)\right).$$

For information nodes, $j = 1$:

$x_{vc_j}^{(l)}$ using $\sum_{i\geq 2}^{d_{\mathsf{vmax}_1}} \lambda_i \sum_{i_1\geq 2}^{d_{\mathsf{vmax}_1}} \lambda_{i_1}$ ·

$g_{mi}\left[f\left(\frac{2}{\sigma_n^2} + (i-1)J^{-1}(x_{cv}^{(l-1)})\right) * g_t\left(\frac{2}{\sigma_n^2} + (i_1)J^{-1}(x_{cv}^{(l-1)})\right)\right]$ and Eq. (4.9)

For parity nodes, $j = 2$:

$$x_{vc_j}^{(l)} = \sum_{i=2}^{d_{\mathsf{vmax}_j}} \lambda_i^{(2)} J\left(\frac{2}{\sigma^2} + (i-1)\, J^{-1}\left(x_{cv}^{(l-1)}\right)\right).$$

end for

Require:

$x_{vc}^{(l)} > x_{vc}^{(l-1)}$.

$\sum_{j=1}^{2} \sum_{i=2}^{d_{\mathsf{vmax}_j}} \lambda_i^{(j)} = 1.$

$\sum_{i=2}^{d_{\mathsf{vmax}_2}} \frac{\lambda_i^{(2)}}{i} = \sum_{i=2}^{d_{\mathsf{cmax}}} \frac{\rho_i}{i}.$

$\sum_{i\geq 2}^{d_{\mathsf{vmax}_2}} \frac{\lambda_i^{(2)}}{i} = \frac{M}{K} \sum_{i\geq 2}^{d_{\mathsf{vmax}_1}} \frac{\lambda_i^{(1)}}{i}.$

Ensure:

$e^{-\frac{1}{2{\sigma_n}^2}} < \frac{1}{\lambda'(0)\rho'(1)}.$

check-nodes which violate the $\rho(x)$. We also exclude check-nodes which are connected to v_{i+1}.

4.4 Simulation Results

We provide the results of optimization and simulations in this section. The matrices used were $\mathbf{Q}_{s1}$ and $\mathbf{Q}_{s4}$, with $p_1 = p_2 = 0.9$ and $p_1 = p_2 = 0.7$, respectively. The optimization was done by fixing

$$\rho(x) = 0.98x^7 + 0.02x^8 \ .$$

Since the convergence constraint is non-linear, a variant of interior-point constrained optimization named sequential quadratic programming (SQP) is used in Matlab. The following degree distribution polynomials were obtained, with the maximum allowed degrees in each class being $d_{vmax_1} = d_{vmax_2} = 15$.

Table 4.1: Optimized Degree Distribution Polynomials

	$\lambda_i^{(1)}$	$\lambda_i^{(2)}$	R	$(E_B/N_0)^*$dB
Q_{s1}	$0.3198x^2 + 0.3468x^{14}$	$0.0816x + 0.2518x^2$	0.5098	0.47
Q_{s4}	$0.3362x^2 + 0.3301x^{14}$	$0.0810x + 0.2527x^2$	0.5181	0.53

We simulated the code for $N = 2048$ for 10 and 20 iterations using Dec-1. To compare the results of a optimization, standard optimized codes for AWGN channels for rate $1/2$ are used as a reference from Table II of [26] with degree polynomials

$$\begin{aligned} \lambda(x) &= 0.23802x + 0.20997x^2 + 0.03492x^3 + 0.12015x^4 \\ &\quad + 0.01587x^6 + 0.00480x^{13} + 0.37627x^{14} \ , \\ \rho(x) &= 0.98x^7 + 0.02x^8 \ . \end{aligned} \tag{4.11}$$

When plotting the results, rate loss was incorporated to compensate for the slightly different rates from Table 4.1. In Fig. 4.2, we show the BER curves for the optimized codes from Table 4.1 and the reference code of maximum variable node degree 15 from Eq. (4.11). The threshold for the degree-15 polynomials from [26] is 0.3347 dB, however this threshold is for sum-product decoding, not the decoding algorithm (Dec-1) presented in the previous chapter. We cannot directly compare the thresholds. This would require extra numerical studies. Instead we directly simulated the degree distribution pairs under the same conditions and compared their performances. The reference curves begin to converge at lower SNRs. However, for both source models, $p = 0.9$ and $p = 0.7$, the optimized curves are steeper.

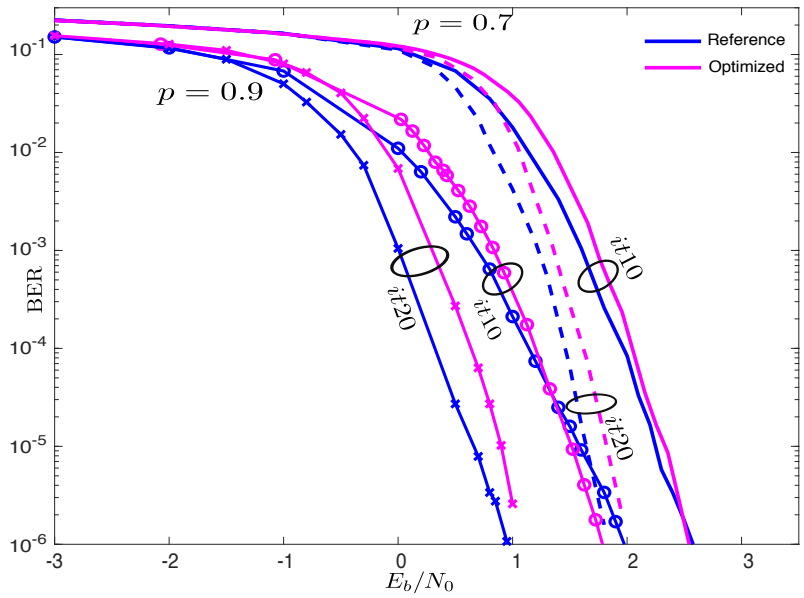


Figure 4.2: Simulation results for codes with similar degree distribution polynomials

For 10 iterations, there is a crossover at higher SNRs, favoring the optimized curves although their convergence started at higher SNRs than the reference curves. After a maximum of 20 iterations, the steepness of the optimized codes is still observable for the source model $p = 0.9$, for $p = 0.7$ it is not. This is explained by looking at the results performed for 10 iterations, where the $p = 0.7$ had a smaller advantage than the optimized curves for $p = 0.9$.

These curves are presented as 'proof-of-concept' for the optimization method. Our intention is to demonstrate that the optimization yields a stronger steepness. The illustrated results for the optimized curves is not showing a performance gain, this however is expected and the reasoning is described below.

The optimization performed in this chapter is a direct counterpart to the decoder architecture described in Chapter 3 with directed edges between information variable nodes. In there, the low entropy source sequence is not compressed before transmission, the system consists of a rate-$1/2$ LDPC code transmitted over an AWGN channel. This is also the code and channel the reference curves are optimized for, meaning the reference curves are able to correct the codewords. The benefit provided by optimizing for the specific decoder structure is in the steepness of the curves, however, for these curves it is not enough to compensate for the convergence point gap between the two codes. This may be an implementation issue limited by the programming language. We provide another result, in Fig. 4.3.

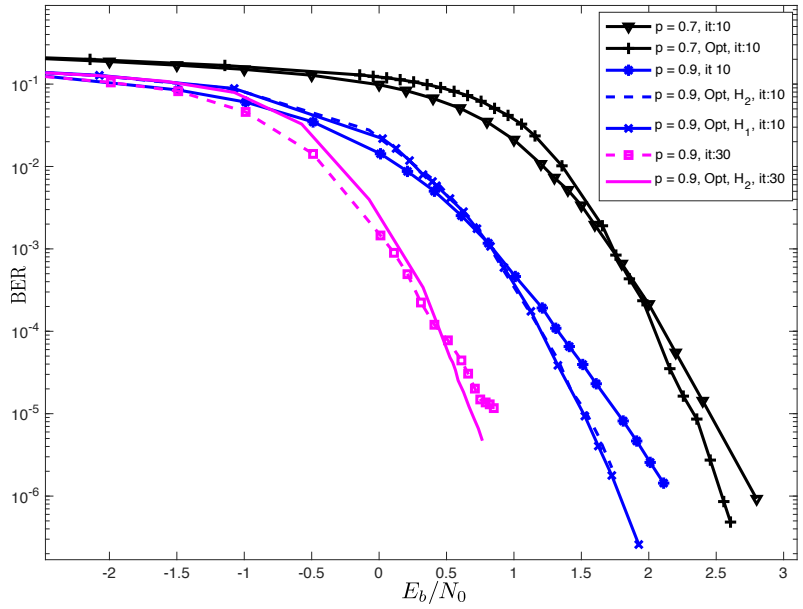


Figure 4.3: Simulation results for codes with similar convergence points

These results for optimized and standard codes for AWGN channels are plotted, for codes which start their convergence at similar SNRs. The standard code is given by Eq. (3.13). Since the maximum variable node degree is 8 for the code from Eq. (3.13) in contrast to the optimized ones which have maximum variable node degree 15, we would expect a flatter performance for the code given by Eq. (3.13) anyway. However, since they start convergence relatively at the same SNR, we plotted them on the same figure. Here, we observe the performance gain also. We conclude from our results that a limitation can arise from the programming platform used for optimization. In the optimization program, we could only simulate a portion of the EXIT curve, a full simulation along the curve did not yield results. The optimization program was able to be completed for a portion of the EXIT chart, otherwise, internally the total maximum number of iterations of the optimizer was exceeded or some other internal parameter was violated. Another disadvantage arises from the Gaussian approximation used. Full density evolution is approximated by the Gaussian assumption to density evolution, which we used on the check node side. Since, the expected gains are in fractions of dBs, these limitations are enough to perturb the final performance of the system. In the presence of such limitations, in practice for the system described above, simulation results should determine the code of choice. At the end of Chapter 3, a JSCC system was described which would compress the source and then perform channel coding on it. Code optimization for such a case would require developments from this chapter.

In Fig. 4.3, the results for the two different $\mathbf{H}$ matrix construction algorithms are also shown.

$\mathbf{H}_2$ is the construction where we restricted neighboring variable nodes being connected to the same check node. No such restriction was in place for $\mathbf{H}_1$. There is almost no difference between the two construction methods. We conclude it is not necessary to limit the construction method to avoid edges between neighboring variable nodes and a specific check-node.

4.5 Summary

In this chapter, a code optimization procedure for a modified LDPC decoder which incorporates source memory is presented. The system we consider has extra edges in the Tanner graph, linking variable nodes. These edges are directed and use intermediate BP estimates to provide extra information to neighboring nodes, which mimics the source model. These links were assumed to be important in designing optimized codes, specifically in the density evolution procedure. The extra edges provide information as a function of the variable node estimates. Meaning the information on these edges are distributed with the same degree polynomial as the information nodes of an LDPC decoder. For optimization this results in a quadratic convergence constraint, which relates to the requirement that the mutual information between decoder and source should increase after every iteration. The results of the optimization program yield degree polynomials which have steeper performances than standard optimized polynomials for AWGN channels. However, the results are subject to the optimization program used, in this case Matlab, and we cannot always compete with standard codes due to underlying numerical and stability (of the optimizer) issues. If the performance of the standard codes are able to overcome the performance gain achieved by optimized ones, then those are a better choice for applications. We do however show the validity of the method by providing performance curves which show the acquired steepness is able to 'catch-up' at high SNRs, even with codes which start their convergence earlier.

In a planned further development, we will use syndrome source compression on the low entropy source sequence and subsequently perform parity computations on the compressed sequence. This would lead to a joint source-channel code, illustrated in Fig. 3.7. Here the source sequence would not be transmitted, hence, there will be no intrinsic information for the source bits. The *left-to-right* link are essential for such a structure and the optimization method in such a case would also require developments from this chapter.

Chapter 5

LDPC Codes in Impulse Noise with Memory

Having covered the treatment of memory in the source sequence using LDPC decoding, we now turn our attention to mitigating the effects of impulse-noise with memory in LDPC decoding. In chapters 3 and 4, we assumed the noise model to be additive white Gaussian. In many practical situations, there is often an impulsive noise component alongside background noise. For example, in power lines, impulsive disturbances are always present. In this chapter, we focus on impulsive noise disturbances and mitigation methods using LDPC decoding.

Impulsive interference can, e.g., be modeled using Middleton's Class-A (MCA) model. For the case with memory, a Markov model is used to describe the dependencies between the noise states. Both states are characterized by white Gaussian noise, with different variances characterizing the background and impulsive noise states. The variance of the impulsive state is then wider than the background state.

For symbol decoding in an impulsive noise environment, it becomes essential to detect which noise state was active during symbol transmission, based on the received samples. This knowledge is used in the iterative decoder. Centrally, the problem is detecting the state sequence of the two noise states. In this chapter, we introduce the MCA model and present decoding steps suitable for an impulsive noise environment. We present three different decoding scenarios and the corresponding results.

5.1 Impulse Noise Modeling

Impulsive interference refers to sporadic noise impulses of short duration and high amplitude.

We differentiate here between two kinds of noise, background and impulsive. Background noise is the interference process always present. This noise is typically modeled as AWGN, caused by resistive noise or some stationary interference.

In contrast, impulse noise is non-stationary. Impulsive interference affects a wide variety of systems, including wireless channels, wireless transceivers in laptops, aeronautical communication systems, etc. The performance of systems affected by impulse noise can be severely hampered if it is not considered during the design process. In order to effectively treat impulsive interference, the first step is to model the noise process effectively. In the next section, we discuss a few impulse noise models. We denote all noise as z.

5.1.1 A Few Statistical Models of Impulse Noise

Impulse noise models can be broadly categorized under empirical models and statistical-physical models. Empirical models arise from measurement data which provide tractable distributions whereas statistical-physical models allow for parametric models where the parameters are different physical attributes of the noise process. Statistical-physical models allow for investigating impulse noise thoroughly by tweaking the parameters. A few of the most commonly used models are MCA, the symmetric alpha-stable ($S\alpha S$) distribution, and the ϵ-mixture model.

5.1.1.1 $S\alpha S$ Model

The $S\alpha S$ model is best described via its characteristic function,

$$\phi(z) = \exp(jvz - \gamma|z|^{\alpha}) \,, \tag{5.1}$$

there is no closed form expression of the noise pdf for this model. Here, α is the characteristic exponent and γ is the dispersion of the distribution around the location parameter v. The $S\alpha S$ model reduces to a Cauchy distribution for $\alpha = 1$ and to a Gaussian distribution for $\alpha = 2$. However, it is not very practical for use in communication systems since the dispersion parameter can be infinite, which is analogous to an infinite variance for a Gaussian distribution.

5.1.1.2 ϵ-Mixture Model

The ϵ mixture model is an empirical mixture model, often expressed as a weighted sum of two Gaussian distributions.

$$p_z(z) = \frac{1-\epsilon}{\sqrt{2\pi\sigma_0^2}} \exp(\frac{-z^2}{2\sigma_0^2}) + \frac{\epsilon}{\sqrt{2\pi\sigma_1^2}} \exp(\frac{-z^2}{2\sigma_1^2}) \ . \tag{5.2}$$

The second term is assumed to describe the impulse noise. The impulsive term can be replaced by any symmetric pdf, i.e., the Laplacian or the Cauchy pdf. ϵ is the probability for impulses to occur.

5.1.1.3 Middleton Class-A Model

One of the most widely used statistical models for impulse noise was introduced by Middleton in [64]. In his 1972 paper, Middleton differentiated between three classes of noise models, class A referring to narrow-band noise [64, 65]. The class-B model is for cases when the noise bandwidth is broader than the receiver bandwidth, and the class-C variant is a mixture of class-A and class-B models.

Middleton differentiated between noise of two origins - man-made disturbances such as electrical transients due to electrical components being plugged into the power line, and environmental noise due to weather phenomena, for example. We have chosen MCA as the impulse noise model we consider for the following. In this section, we present a basic introduction to the Middleton Class - A model.

We begin by re-introducing the received signal samples,

$$y_n = x_n + z_n \ . \tag{5.3}$$

Here, y_n is the n^{th} received value for a bit of the codeword of length N. z_n is the noise sample added to the n^{th} transmitted bit of the codeword $\mathbf{x}$. z_n consists of two components, the background Gaussian noise z_G and the impulsive interference denoted as z_I.

According to the MCA model, impulsive sources are assumed to be independently distributed in time and space according to a Poisson distribution [64]. The other wave parameters, i.e., envelopes, frequencies, and phases of the noise source emissions are randomly distributed.

The PDF of the amplitude of a noise sample z_n is given by

$$\begin{aligned} p(z_n) &= \sum_{m=0}^{\infty} P(m) \cdot \mathcal{N}(z_n; 0, \sigma_m^2) \\ &= \sum_{m=0}^{\infty} P(m) \cdot \frac{1}{\sqrt{2\pi\sigma_m^2}} \exp(\frac{-z^2}{2\sigma_m^2}) \; ; \end{aligned} \tag{5.4}$$

where z_n is a sample from the Gaussian PDF with $zero$ mean and variance σ_m^2.

We observe that the noise PDF is a Gaussian mixture model with an infinite number of terms in the summation. m refers to the number of impulsive sources. We now define a parameter A known as the impulsive index. A is the density of impulsive events in a certain observation period, it can also be described as the duty cycle of the impulse noise.

$$A = \frac{T_D}{T_I} \; , \tag{5.5}$$

where T_D is the duration of impulsive events and T_I is an observation period. Hence, A can be seen as a rate of impulse noise events. Given that different impulsive sources are active independent of each other but with an overall activation rate of impulsive disturbance A, $P(m)$ can now be defined as an independent Poisson point process,

$$P(m) = \frac{A^m \cdot e^{-A}}{m!} \; . \tag{5.6}$$

By definition, $A \leq 1$ since it is defined as a density within an observation period. In Fig. 5.1, we show the pulsed appearance of the MCA noise model for a noise burst. When the noise does not occur in bursts, $T_D = \eta \cdot \tau$, where τ is the width of noise pulse and η is the total number of noise emissions in T_I. Middleton limits the value of A to the range $[10^{-6}, 1]$. Equation (5.4) can be approximated by a finite number of terms [65]. For our purposes, it

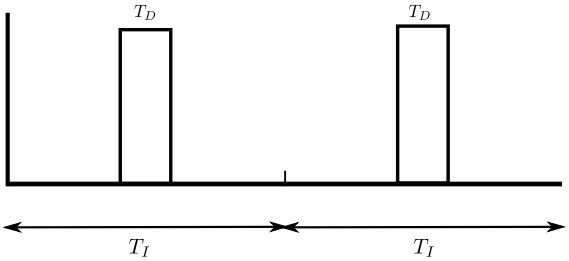


Figure 5.1: Pulsed appearanace of MCA noise

is sufficient to limit the summation to two terms, distinguishing between two states, $m = 0$

signifying the background AWGN state and $m = 1$ describing an impulsive state.

The variances σ_m^2 are given by

$$\sigma_m^2 = \sigma_G^2(1 + \frac{m_n}{A\Gamma}) \; ; \;\; m = \{0, 1\} \; , \tag{5.7}$$

where, σ_G^2 is the variance of the background Gaussian noise and the subscript n in m_n refers to the n^{th} bit. Γ, known as the Gaussian factor, represents the power ratio between the background AWGN and impulsive noise,

$$\Gamma = \frac{\sigma_G^2}{\sigma_I^2} \; . \tag{5.8}$$

The noise PDF of z_n is then given as

$$P(z_n) = \frac{\alpha_0}{\sqrt{2\pi\sigma^2_{m_n=0}}} \exp(\frac{-z_n^2}{2\sigma^2_{m_n=0}}) + \frac{\alpha_1}{\sqrt{2\pi\sigma^2_{m_n=1}}} \exp(\frac{-z_n^2}{2\sigma^2_{m_n=1}}) \tag{5.9}$$

with

$$\alpha_0 = \frac{A^0 \cdot \exp(A)}{0!} \quad \text{and} \;\; \alpha_1 = \frac{A^1 \cdot \exp(A)}{1!} \; . \tag{5.10}$$

We notice that the MCA model, when approximated by two terms, is identical to the ϵ-mixture model using Gaussian densities, where, ϵ is the probability of impulses.

We have assumed a BPSK modulated transmission scheme. A symbol period is denoted as T_e. If the impulse noise duration, T_D is less than or equal to the symbol duration T_e, the MCA noise is considered memoryless, i.e., the noise affecting neighboring pulses of the transmit signal are independent. Hence, when decoding the received signal, neighboring variable nodes or symbols do not provide any information about each other beyond the constraints imposed by the LDPC parity-check matrix $\mathbf{H}$.

If the impulsive duration, T_D is greater than the symbol period T_e, then

$$D = \left[\frac{T_D}{T_e}\right] \tag{5.11}$$

symbols (rounded) within the impulsive duration are affected by the same noise state, $m_n = ... = m_{n-D-1}$. Then, the pdf of the MCA density can be written as

$$p(z_n) = \sum_{m_n \in \{0,1\}} p(m_n) P(z_n | m_n). \tag{5.12}$$

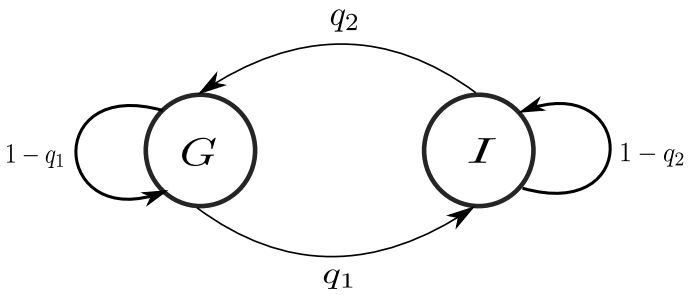


Figure 5.2: 2-state Markov model for MCA noise

This dependency can be reformulated as a Markov model as per [64, 65]. We formulate a two-state Markov model with the probabilities $q_1 = P(m_n = 1|m_{n-1} = 0) = P(I|G)$ and $q_2 = P(m_n = 0|m_{n-1} = 1) = P(G|I)$. The impulse-noise model then becomes similar to the Gilbert-Elliott channel model with the Gaussian noise state signifying the better channel and the impulsive state signifying the bad channel. We derive the transition probabilities q_1 and q_2 based on previously chosen parameters, D and A.

For a Markov model, the average time to stay in a state can be calculated. The average time to stay in the impulsive state is

$$D = \sum_{i=0}^{\infty} 1 \cdot (1-q_2)^i = \frac{1}{1-(1-q_2)} = \frac{1}{q_2} \,. \tag{5.13}$$

This duration is the average 'burst length', viewed as a noise burst with variance higher than the background noise of the channel. Similarly, the average time spent in the Gaussian state can be calculated as $1/q_1$. This can be termed the average 'gap length', the gap between switching to impulsive states, i.e., the number of symbols affected only by Gaussian background noise, on average.

The number of symbols unaffected by impulse noise is, according to Fig. 5.1:

$$\begin{aligned} \frac{1}{q_1} &= \frac{T_I - T_D}{T_e} \\ &= \frac{T_I}{T_D} \cdot \frac{T_D}{T_e} - \frac{T_D}{T_e} \\ &= \frac{D}{A} - D \,. \end{aligned} \tag{5.14}$$

Using eqs. (5.13) and (5.14), we can write the order-1 Markov transition probability matrix, $\mathbf{Q}$ which describes the memory of the process. When, $T_D = T_e$, this model describes the memoryless case.

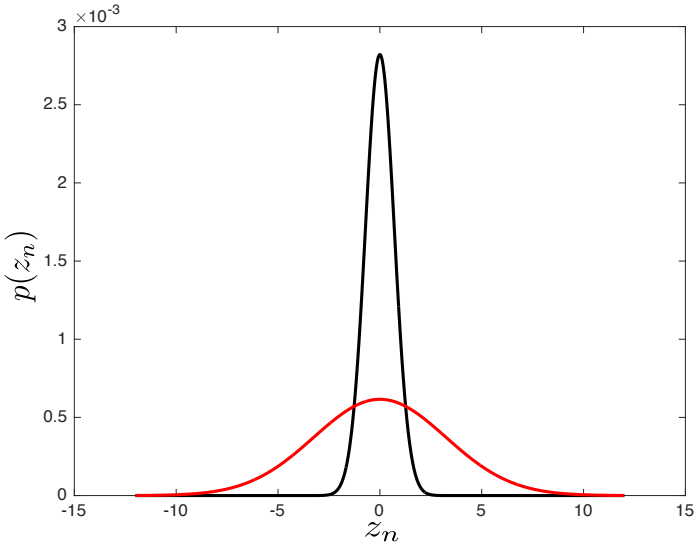


Figure 5.3: MCA noise model using 2-terms

$$\mathbf{Q} = \begin{bmatrix} P_{G\to G} & P_{I\to G} \\ P_{G\to I} & P_{I\to I} \end{bmatrix} = \begin{bmatrix} (1-q_1) & q_1 \\ q_2 & (1-q_2) \end{bmatrix} ;$$

where $\mathbf{G}$ represents the background Gaussian noise and $\mathbf{I}$ represents the impulsive noise state.

In Fig. (5.3), we show the two-term MCA model from Eq. (5.9). The Gaussian density plotted in red corresponds to the impulse noise pdf, and the density plotted in black corresponds to the background noise pdf. This figure is plotted for $A = 0.5$ and $\Gamma = 0.1$. The value of D is set to 10.

From the figure, we note that there is a crossover point between the two densities where the probability of a noise sample being generated by either the background Gaussian or the impulsive Gaussian is equal. This threshold value can be used as a discriminant for deciding which noise state gives rise to the considered noise sample. Hence, we define a threshold value at

$$P(z_n|m_n = 0) = P(z_n|m_n = 1) \, . \tag{5.15}$$

$$\frac{\alpha_0}{\sqrt{2\pi\sigma_0^2}} \exp(\frac{-z_n^2}{2\sigma_0^2}) = \frac{\alpha_1}{\sqrt{2\pi\sigma_1^2}} \exp(\frac{-z_n^2}{2\sigma_1^2}) \, ,$$
$$z_n^2 = \frac{2\sigma_1^2\sigma_0^2}{\sigma_1^2 - \sigma_0^2} \cdot \log(\frac{\alpha_0 \cdot \sigma_1}{\alpha_1 \cdot \sigma_0}) \, .$$

Thus, the threshold value is set to

$$z_o = \pm\sqrt{\frac{2\sigma_1^2\sigma_0^2}{\sigma_1^2 - \sigma_0^2} \cdot \log(\frac{\alpha_0 \cdot \sigma_1}{\alpha_1 \cdot \sigma_0})} \,. \tag{5.16}$$

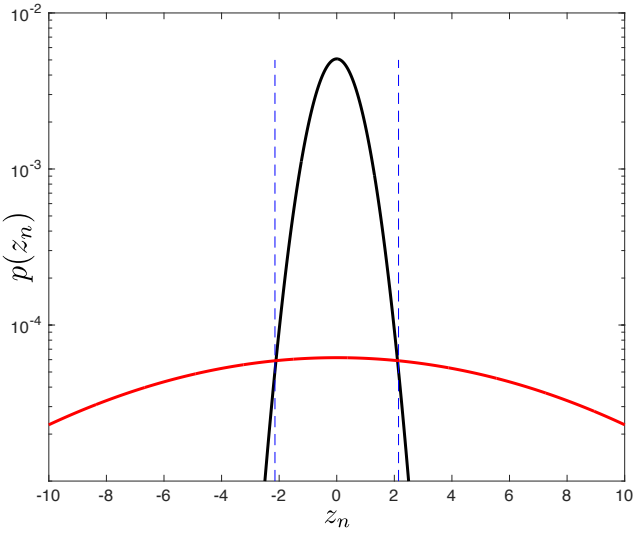


Figure 5.4: Threshold drawn at the noise sample value at which both densities are equiprobable

In Fig. 5.4, we see that the threshold value is drawn at the noise sample where there is a crossover between the probabilities of the constituent densities. The figure was generated for $D = 2,\ A = 0.1,\ \Gamma = 0.2$.

In the next section, we describe the different decoding methods used for mitigating impulse noise with memory.

5.2 Decoding Methods

We used BP decoding for an LDPC code using LLRs. From Eq. (2.34), we know that LLRs for LDPC codes can be factored into three summands, intrinsic, a-priori, and extrinsic LLR. The intrinsic LLR expresses the channel properties, i.e., the noise environment of the system. Hence, the noise state estimation will affect the intrinsic LLR, L_{itr}, whose computation is shown subsequently.

5.2.1 Intrinsic LLR Computation based on Weighted Sums of Different State Distributions

A simple way to compute L_{itr} is to compute $P(y_n|x = \pm 1)$ for both noise densities $\mathcal{N}(0, \sigma_G)$ and $\mathcal{N}(0, \sigma_I)$ and sum the values based on the weights of the two distributions. The weighted sums take into account the average behavior of the noise process.

The resulting intrinsic LLR is given by

$$L_{itr}(y_n) = \frac{(1-A)P(y_n|+1, m_n = 0) + (A)P(y_n|+1, m_n = 1)}{(1-A)P(y_n|-1, m_n = 0) + (A)P(y_n|(-1, m_n = 1)} \,. \tag{5.17}$$

This computation method does not take into account the memory in the noise process, rather it reflects the steady-state probabilities of the Markov model since the weighting is based on those values. For the 2-state model, this method is similar to the computation given in [66].

We take this method as a benchmark for comparisons in our simulations.

5.2.2 Noise State Estimation via Viterbi Algorithm

For estimating the sequence of noise states, we now implement a Turbo-like structure with an 'outer' trellis-based Viterbi decoder estimating the noise states and an 'inner' LDPC decoder calculating the BPSK transmitted symbols.

We have described the noise process as an order-1 Markov model. Viterbi decoding can be used to estimate the state sequence of a Markov model. A detailed description of the Viterbi decoder is given in [4]. Since the sequence of noise states is governed by the transition probabilities of the Markov model, we incorporate the transition probabilities, given by $\mathbf{Q}$ in the state transitions of the Trellis.

Let us consider the annotated path $\mathbf{I} \to \mathbf{I}$ on the trellis. The received value y_n is generated by the state on the left of the path. To find the probability of y_n being generated by the left state $\mathbf{I}$, we use the Gaussian pdf to find the emission probability of y_n. The probability for the next state (the state on the right side of the path segment) to be $\mathbf{I}$ is then a product of the Gaussian probability we have calculated and the transition value $1 - q_1$. All the other Trellis path segments are calculated similarly. The mean of the Gaussian noise processes are taken to be the hard decision values obtained from the variable nodes in the Tanner graph at the current iteration, $\tilde{y}$. The sequence of states estimated by the Viterbi decoder are in turn used in calculating the intrinsic LLR using $\frac{2 \cdot y}{\sigma_n^2}$, from Eq. (2.37). This interplay between the Viterbi decoder and the LDPC Tanner graph is shown in Fig. 5.5.

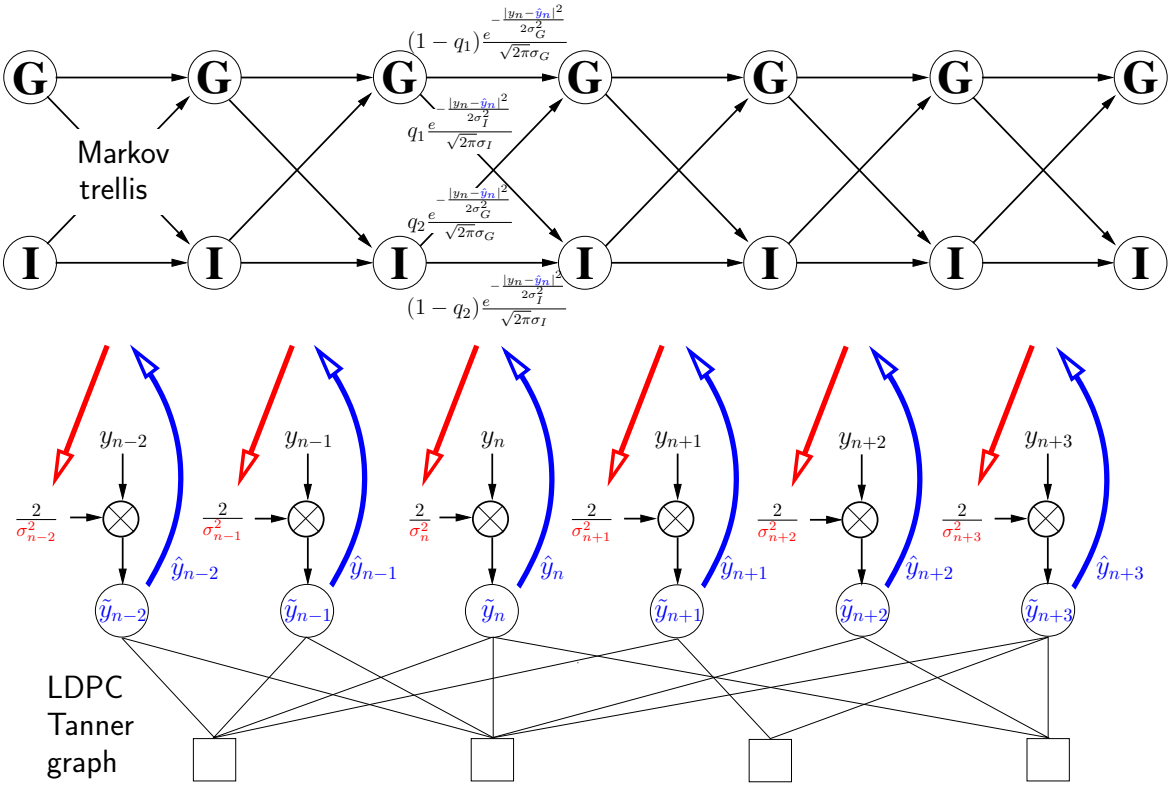


Figure 5.5: Noise state estimation using a trellis

Unlike the previous method, the memory in the noise process is accounted for by using Viterbi decoding to calculate the noise state sequences. This is our proposed method for distinguishing between whether impulsive or background Gaussian noise was present during transmission.

In simulations, Viterbi in log-domain was used since it is more numerically stable than using only probabilities. The accumulated metric in the Viterbi algorithm (Alg. 1) in log domain replaces the multiplication of Eq. (2.11) with an addition of logarithms. In [67], a similar method was investigated, however, the contribution described here was arrived at independently.

5.2.3 Threshold Estimation

The third method we use for noise state detection uses the threshold described in Eq. (5.16). The threshold, shown in Fig. 5.4, provides us with a boundary region around the the mean, within which the probability of the received value to be transmitted during a time window when the background noise process was only present is higher than that of the impulsive process. Outside this region bounded by the threshold, the probability for the impulsive noise state is higher.

In order to use Eq. (5.16), we first do a hard decision for the transmitted BPSK symbol by slicing at zero. A hard decision, $\hat{x}_n$, is obtained as the sign of the received value. After the hard decision, we apply the thresholding operation shown in Eq. (5.18) and decide the value of the noise state, m_n. We then use $L_{itr} = 2 * y_n/\sigma^2_{m_n}$, from Eq. (2.37), for calculating the intrinsic LLR. This method also does not update the intrinsic LLR during LDPC decoding, and also does not take memory into account.

$$p_Z(y_n|X_n = \hat{x}_n) \approx \begin{cases} p_Z(y_n - \hat{x}_n|m_n = 0), & \text{if } |y_n - \hat{x}_n| \leq z_0 \ , \\ p_Z(y_n - \hat{x}_n|m_n = 1), & \text{otherwise} \ , \end{cases} \tag{5.18}$$

where, the threshold value z_0 was given in Eq. (5.16).

The threshold can be used to simplify the intrinsic LLR computation given by Eq. (5.17),

$$p_Z(y_n|X_n = \pm 1) \approx \begin{cases} p(m_n = 0)p_Z(y_n - x_n|m_n = 0), & \text{if } |y_n - x_n| \leq z_0 \ , \\ p(m_n = 1)p_Z(y_n - x_n|m_n = 1), & \text{otherwise} \ , \end{cases} \tag{5.19}$$

When Eq. (5.19) is used, it allows for different noise states in the numerator and the denominator. Thus, only two exponents need to be computed instead of four, in Eq. (5.17).

5.3 Results and Discussion

In this section, we provide BER simulation results comparing the performance of the three detection methods described in the previous section.

We have simulated a rate-$1/2$ length 2048 LDPC code with degree distribution polynomials

$$\begin{aligned} \lambda(x) &= 0.28286x + 0.39943x^2 + 0.31771x^7, \\ \rho(x) &= 0.6x^5 + 0.4x^6. \end{aligned}$$

The LDPC decoder is run for a maximum of 20 iterations for all three alternatives. We have computed BERs for three different Gaussian factors (Eq. (5.8)) $\Gamma = 0.1, 0.01,$ and 0.001. For each value of Γ and three different values of $D = 2, 6,$ and 10 (Eq. (5.11)) we obtain different transition matrices, $\mathbf{Q}$. The impulsive index A (Eq. (5.5)) was set to 0.1 for all simulations. Equations (5.13) and (5.14) illustrate how to obtain the Markov transition matrix $\mathbf{Q}$ from the parameters A and D.

$$\mathbf{Q}_{D=2} = \begin{bmatrix} 0.9444 & 0.5 \\ 0.0566 & 0.5 \end{bmatrix} \tag{5.20}$$

$$\mathbf{Q}_{D=6} = \begin{bmatrix} 0.9815 & 0.1667 \\ 0.0185 & 0.8333 \end{bmatrix} \tag{5.21}$$

$$\mathbf{Q}_{D=10} = \begin{bmatrix} 0.9889 & 0.1 \\ 0.0111 & 0.9 \end{bmatrix} \tag{5.22}$$

We notice that for $D = 2$, the second column of the matrix has a value of 0.5 for both rows, meaning there is no memory in the noise process, the system switches to the impulsive state from both the Gaussian or impulse noise state with equal probability. For $D = 6$ and $D = 10$, there is memory in the process.

In Fig. 5.6, the BER curves for $D = 2$ are plotted. We used Eq. (5.18) for the threshold method. We observe that when there is no memory in the impulse noise process the 'weighted sums' method from Section 5.2.1 performs the best, followed by the Trellis and the threshold estimation methods from sections 5.2.2 and 5.2.3, respectively. We also notice that the lower the value of Γ, the better the performance. For lower values of Γ, the impulse noise variance is significantly higher than the background Gaussian, the noise states become easier to distinguish for all three methods, which accounts for the progressive improvement shown in Fig. 5.6. However, with increases in Γ, the variances of the background and impulsive states become similar, as given by Eq. (5.7). For $\Gamma = 100$, σ_0^2 and σ_1^2 are almost identical, hence $\Gamma = 100$ should emulate the impulse noise free case. From the blue curve in Fig. 5.6, this is confirmed. In figs. 5.7 and 5.8, the BER curves for $D = 6$ and $D = 10$ are plotted, respectively. For higher values of D, i.e., when the average burst length is longer, as expected, the trellis estimation method provides the best performance since it estimates the noise state sequence. The other two methods cannot estimate the memory, their performance suffers.

We observe from our results that the BERs are in the order of $[10^{-3}, 10^{-4}]$. When impulse noise is present in a transmission scheme, the observed flooring effect is expected. This is due to the fact that E_s/N_0 relates to the Gaussian background noise, only. The impulse noise is constant in its strength and presence. A further reduction in BER can, e.g., be obtained by an outer Reed-Solomon code.

In Fig. 5.9, BER curves are plotted showing the trellis method for the three values of the impulsive length D for $\Gamma = 0.001$. We note that degradation of performance with increasing

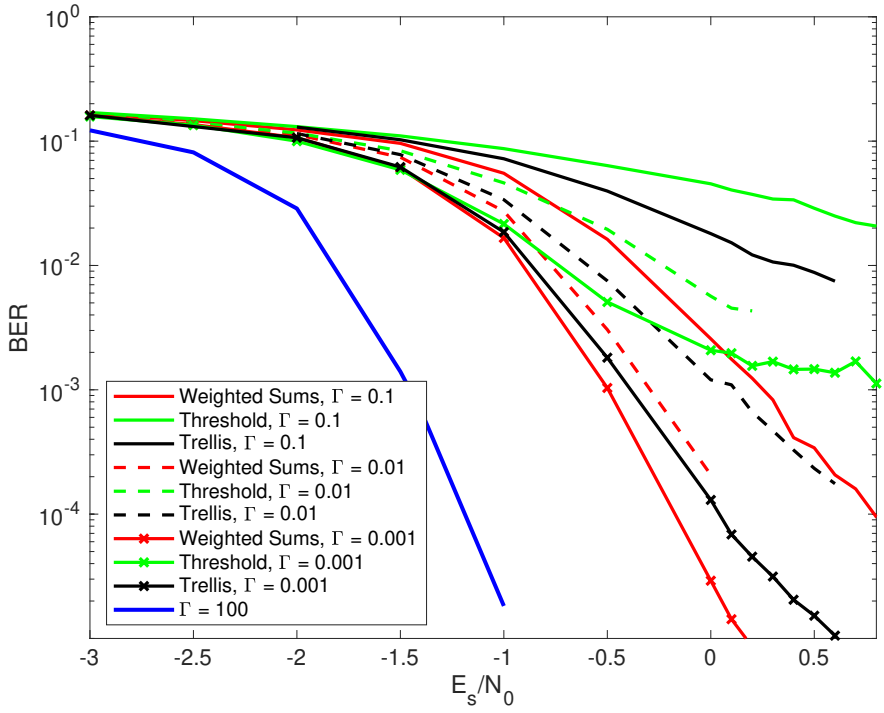


Figure 5.6: BER curves for $D = 2$

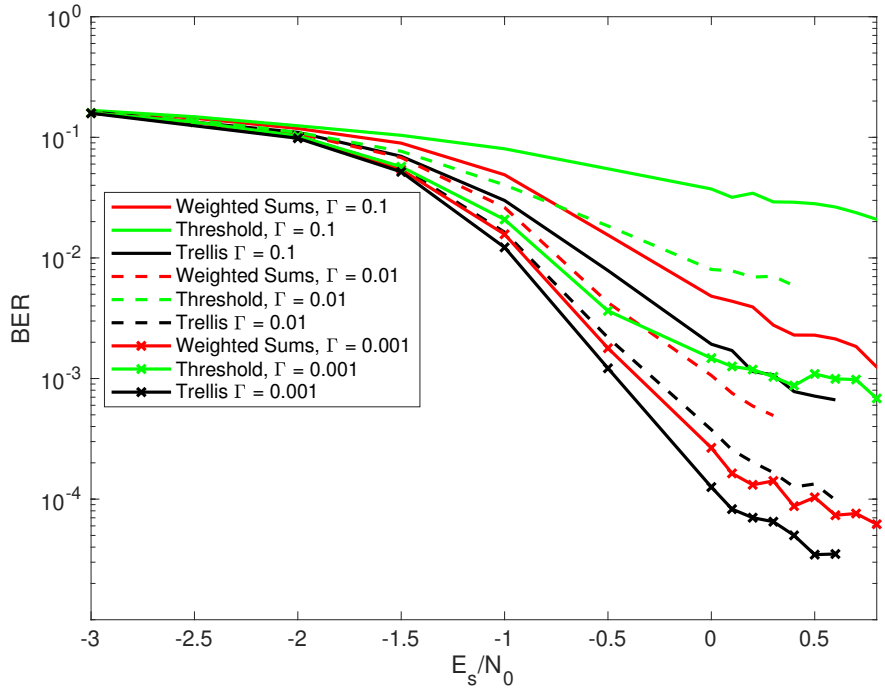


Figure 5.7: BER curves for $D = 6$

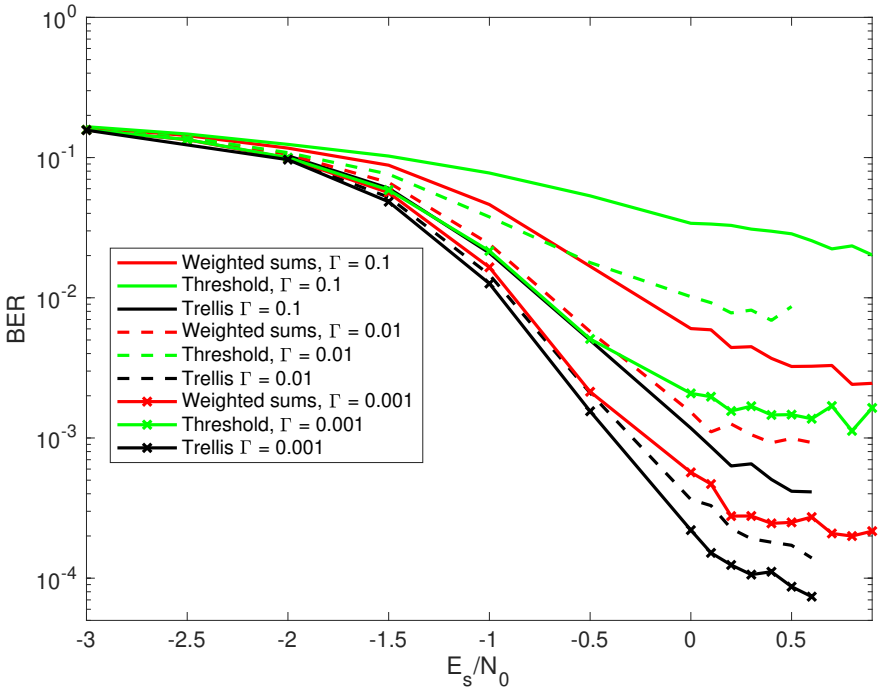


Figure 5.8: BER curves for $D = 10$

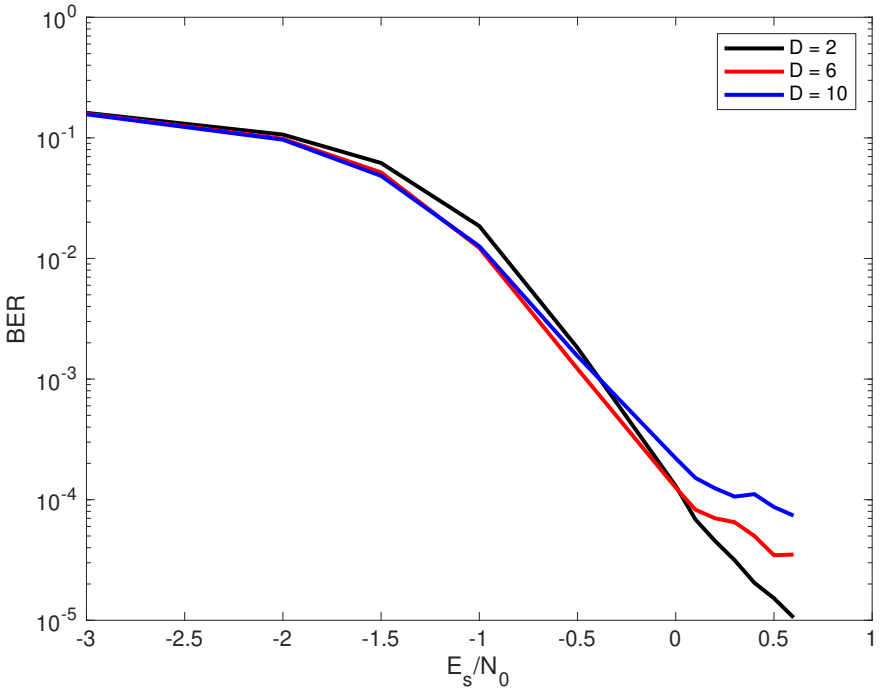


Figure 5.9: BER curves showing the effect of memory using the trellis based method

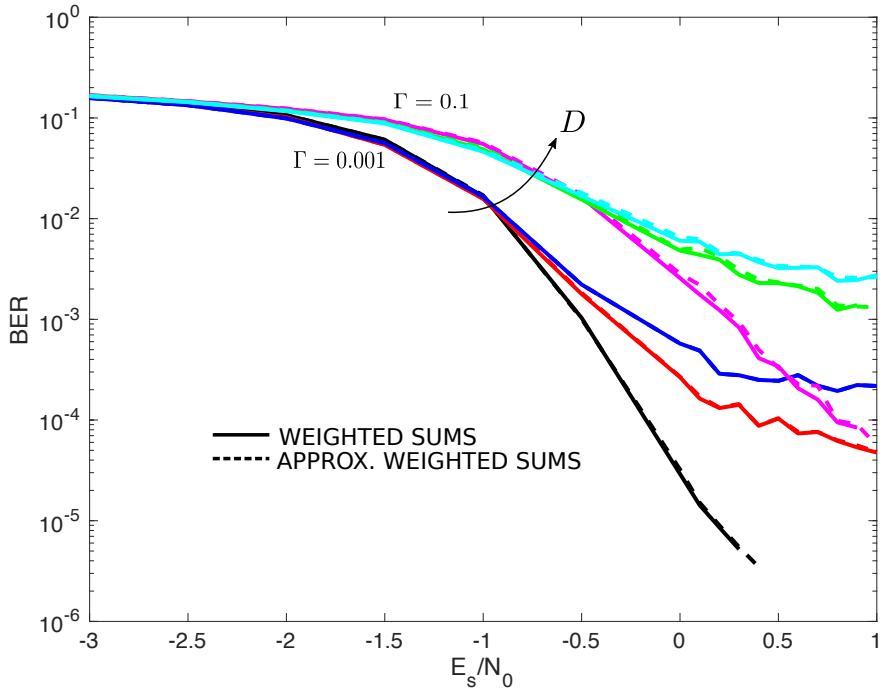


Figure 5.10: BER curves for approximating the Weighted Sums approach using the computed threshold for values of D = 2, 6, and 10

memory of the impulsive process. In Fig. 5.10, the approximation to the weighted sums method of Eq. (5.17) by Eq. (5.19) is given. The approximation yields almost identical performance curves, which means that the approximation can be used in place of the weighted sums method.

5.3.1 Discussion on Simplifying the Trellis-based Method

We now turn our attention to simplifying the trellis-based decoder for impulse noise with memory. In Chapter 3, we simplified the trellis based Turbo coding structure by using a one-step approximation of the trellis inside the Tanner graph. Similarly here, in the Tanner graph, we were also investigating if the left node of a variable node could likewise provide information to the right node, and avoid the Turbo like inner and outer decoder structure.

The goal is to iteratively compute the intrinsic LLR, using intermediate Tanner graph results. The iterations of the LDPC decoder provide probabilities of the BPSK transmitted symbols. In order to use the calculated BPSK symbol probabilities to update the noise state estimation, there are a few steps involved. Our goal would be to estimate $P(m_n = 0|y_n)$ and $P(m_n =$

$1|y_n)$. Let us further simplify.

$$\begin{aligned} P(m_n|y_n) &= P(y_n|m_n) \cdot P(m_n) \cdot \frac{1}{P(y_n)} \\ &= P(y_n|m_n) \sum_{m_{n-1}} (P(m_n|m_{n-1})P(m_{n-1})) \cdot \frac{1}{P(y_n)} \\ &= P(y_n|m_n) \cdot P(m_n|m_{n-1}) \cdot \frac{1}{P(y_n)} \ , \quad m_n \in \{0,1\}. \end{aligned} \tag{5.23}$$

The value for $P(m_n|y_n)$ is dependent on the noise state at the left node indexed by $n-1$, to incorporate the memory from only the previous node. In essence, we are simplifying the complete trellis decoding to a two path computation per segment of the trellis. For such an estimation of the noise state at variable node n, there are two components to consider, as shown in Eq. (5.23). The first component uses the received value at the n^{th} node while the second component accounts for the dependency or memory from the left node. $\frac{1}{P(y_n)}$ is a constant factor for the computation for both $m_n \in \{0,1\}$. In order to compute $P(y_n|m_n)$ we use the intermediate LDPC decision $\tilde{y}_n$, similar to the trellis based method. Also for deciding the state at node $n-1$ using the threshold method according to Eq. (5.18), $\hat{x}_{n-1} = \tilde{y}_{n-1}$ is used. As we have seen from our plotted results, using the threshold method does not provide good estimates of the noise states when there is memory present, hence this simplified method does not yield promising results.

Without deciding the noise state m_{n-1}, as shown in Eq. (5.23), if we had computed path probabilities for both noise states at node n-1, the number of path computations per segment of the trellis would be equal to the trellis-based method. This would not be a simplification computationally as well as leading to inferior solutions since we would ignore the sequence memory. When we decide the state at node $n-1$ as shown in Eq. (5.23), we obtain probabilities for $P(m_n = 0|y_n)$ and $P(m_n = 1|y_n)$, we decide the noise state at node n to be the one which has higher probability. This is another hard decision, this combined series of hard decisions leads to inferior performance. We conclude that simplifying the complete trellis estimation does not work for impulse noise. Thus, updates to the L_{itr} directly during BP iterations in the Tanner graph are not possible.

5.4 Summary

In this chapter, we have investigated the MCA impulse noise model with memory. In practical systems, depending on the parameters, impulse noise may affect consecutive symbols. This noise corruption can be expressed as a Markov model since the impulse noise affecting adja-

cent symbols has memory, i.e., dependency between the symbols. Here, we have described this memory as a first-order Markov model.

We then investigated a Turbo-like scheme in which a Viterbi decoder estimates the noise state sequences of the Markov model and an LDPC decoder uses the state information provided for symbol decoding. This is our proposed approach, and we see from the results provided that the method performs well at detecting impulse noise states. We also investigated a threshold detection scheme and a weighted sums approach for detecting the noise states based on the average behavior. Since neither of these two methods can make use of the memory, the trellis-based approach has a clear advantage. For the trellis-based approach, a Turbo-like decoding scheme is required to pass information iteratively between the LDPC decoder and the Viterbi decoder. We looked into possibilities of simplifying this Turbo structure to obtain direct forwarding in the Tanner graph from left-to-right variable nodes for making use of the noise memory. However, this problem cannot be solved successfully, since in order to forward from left to right nodes, we would need to estimate the noise state on the left node and hence, we are left with a recursive problem. Thus, the trellis-based decoding approach is finally the best method for detecting impulse noise with memory.

Chapter 6

LDPC based Decision-Feedback Equalization

In this chapter, we turn our focus to incorporating equalization into LDPC decoding. Equalization at the receiver is a 'pre-processing' step required to cancel inter-symbol interference that arises from transmit pulses being spread in time during channel transmission. Due to this, adjacent transmit symbols result in received pulses which are a combination of several symbols. Equalization techniques are therefore required to treat the received pulses such that this spreading can be canceled, and decoding is then performed. There are many types of equalization techniques, Turbo equalization being one of them in which equalization and decoding are performed iteratively with the results of one process acting as a-priori information for the other.

In this chapter we have performed equalization using intermediate values from the message passing decoding inside the Tanner graph. We use the variable node LLRs, which yield estimates of the transmitted BPSK symbols, for equalization, leading to an iterative procedure which uses intermediate decoding results to cancel interference from the adjacent symbols. Our results show that our integrated method performs better than sequential equalization and decoding and thus is a competitor for sequential methods, while having lower complexity than Turbo equalization.

6.1 Basics of Equalization

Equalization is the process of canceling inter-symbol interference (ISI) caused by a transmission channel to the transmitted signal. The transmitted message, a sequence of numbers from a finite filed, GF(m) is mapped to a waveform according to a chosen modulation scheme. Mostly, transmission channels are assumed to be linear time invariant (LTI) systems. During transmission over the channel, this signal pulse is spread in time and distorted by noise such that at the receiver, the corresponding received pulse is a linear combination of a few transmitted symbols, i.e., results in ISI. For accurate reconstruction of the transmitted waveform at the receiver, the ISI must be canceled.

For equalization, the channel transfer function must be known. Channel coefficients can be obtained prior to transmission via the transmission of training sequences. The channel transfer function can be adaptively estimated during information transmission, too. Both approaches are used in practice. The latter being necessary for channels which vary over time. Adaptive equalization can also be combined with the initial measurement phase in some instances. Regardless of how the channel transfer function coefficients or equalizer coefficients are obtained, there are different equalizer structures. We provide brief descriptions below.

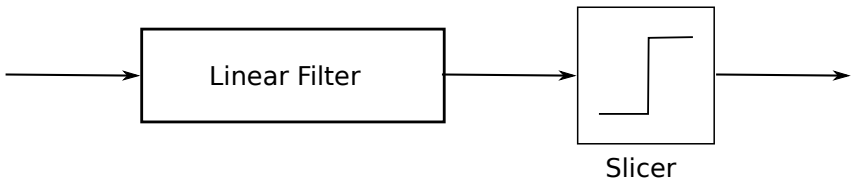


Figure 6.1: Linear equalizer

A linear equalizer, shown in Fig. 6.1, consisting of a linear filter and a slicer, was the first equalizer structure to be extensively studied. The two principle methods used for optimizing the coefficients of the linear filter were the zero-forcing (ZF) criterion, and minimizing the mean squared error (MMSE) between the equalizer output and the channel input. Zero-forcing forces the overall impulse response to be a delta function, with zero neighboring values. Nyquist, in 1928, delivered sufficient conditions for zero-forcing in a noise-free environment. Later developments generalized these conditions for different channel models and waveforms. While ZF aims at eliminating ISI at the sampling instant, this comes with a trade-off that noise may be amplified, since the ZF criterion does not consider the noise at all. In contrast to this method, optimizing the received linear filter coefficients using an MMSE criterion aims at minimizing the total power of the ISI and noise, i.e., maximizing the SNR on average, at the sampling instant.

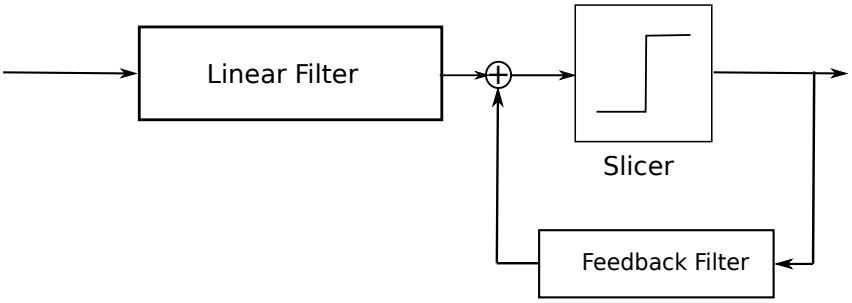


Figure 6.2: Decision-feedback equalizer

The next equalizer structure we discuss is the Decision Feedback Equalizer (DFE) shown in Fig. 6.2. DFE uses a recursive algorithm to cancel the interference of previous symbols, assumed correctly decided. The structure consists of a linear feed-forward (FFF) filter and a feedback filter (FBF). The minimum-phase property to avoid pre-cursors required for DFEs is obtained by using a whitened matched filter as the feed-forward filter, the impulse response of which consists of the cascade of the transmitter, the channel, and itself. The output after the feed-forward filter then only consists of post-cursor components. Assuming that previous symbols have been correctly decided, a feedback filter subtracts the post-cursor components from previous symbols, and a slicer makes the decision on the current symbol. Incorrect estimation of symbols can lead to error propagation when using a DFE structure. In order to avoid this problem, the feedback filter can be moved to the transmission side - this method is called pre-coding. The principle of pre-coding uses channel state information (CSI) on the transmit side to mitigate ISI on the receiver side. Tomlinson-Harshima precoding, shown in Fig. 6.3, is a frequently used solution for equalization when a duplex channel is available.

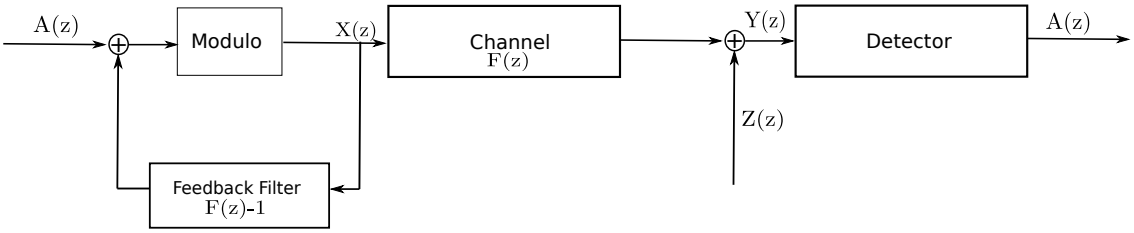


Figure 6.3: Tomlinson-Harashima precoding

We observe from Fig. 2.1 that encoding for forward error correction and the convolution with the channel matrix during transmission are performed serially. Hence, on the receiver side, equalization and decoding are also performed serially. This serial concatenation has led to turbo-like equalization techniques as a possible solutions.

We have discussed Turbo-like solutions in both the previous chapters. Iterations between the

'inner' and 'outer' decoders leads to the two disjoint processes of equalization and decoding iteratively exchanging information in order to decode the transmitted symbols. We show an illustration of a possible serial turbo-equalization process in Fig. 6.4, following [68]. Here, on the receiver side, we start with the equalization process first, since it is the 'outer' process. $L_C(C_n)$ is the LLR value for the received channel output, only the MAP equalizer has access to channel outputs. $L(X_n)$ is the a-priori value obtained by interleaving the extrinsic information obtained from the MAP decoder. In the first iteration, $L(X_n)$ is zero since the MAP decoder has not performed any iterations yet. The extrinsic information output from the equalizer is used as 'intrinsic + a-priori' information for the MAP decoder. In this realization, the MAP decoder has no access to the channel and thus only uses the equalizer output.

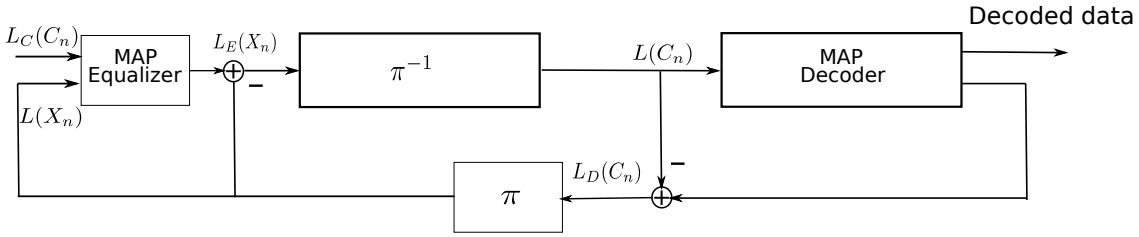


Figure 6.4: Turbo equalizer

6.2 Incorporating Equalization into the Tanner graph

The Tanner graph, as shown in Fig. 2.5, illustrates the LDPC code structure. When there is no interleaver between LDPC encoding and transmission, neighboring variable nodes of the Tanner graph represent serially transmitted symbols. Then ISI also distorts neighboring variable nodes. Recalling our description of a minimum-phase channel impulse response, we deduce that the ISI on a variable node v_q is due to its post-cursors only, i.e., nodes $v_{q-1}, ..., v_0$. Canceling the ISI on a node v_q then requires symbol estimates provided by the nodes to the left of it. We have already published some results in this direction in [69].

6.2.1 Joint LDPC-decoding and Equalization

The intrinsic LLR from Eq. (2.37) reflects the channel information. Using intermediate symbol estimates from variable nodes of the LDPC decoder, equalization is performed on the received analog values, and then subsequently the intrinsic LLR computation is updated. This proposed iterative equalization scheme updates the intrinsic LLR after every iteration in the Tanner graph. This method is similar to a decision feedback equalizer, the 'decision' being the intermediate results of the SPA, thus making the process iterative as well. In Fig. 6.5, we show our proposed method.

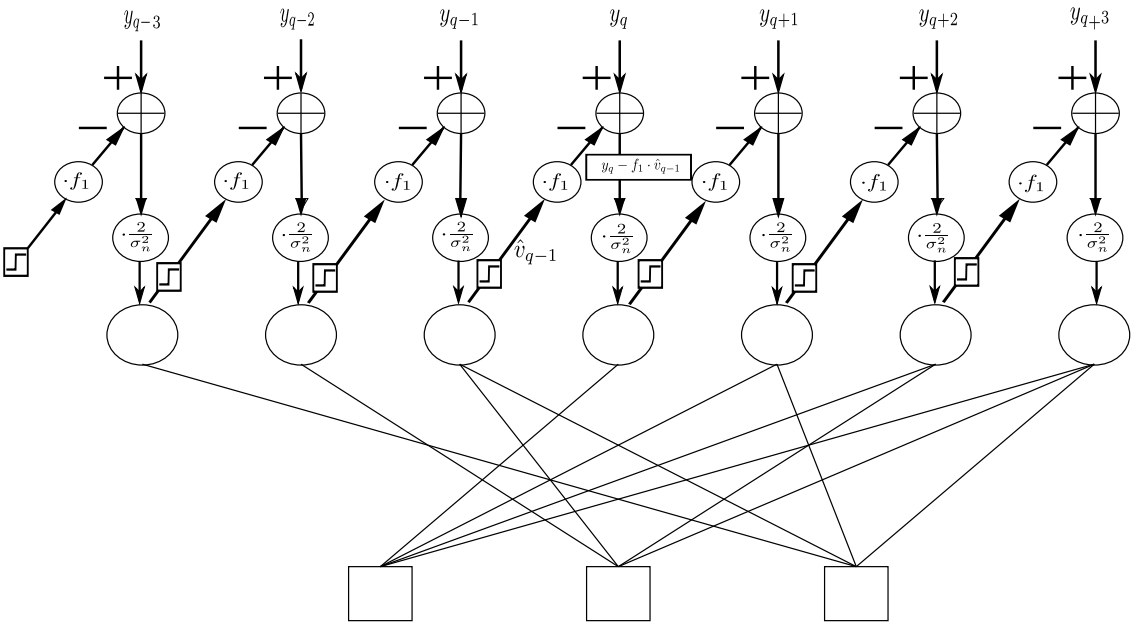


Figure 6.5: Tanner graph for joint LDPC decoding and equalization

Let us assume a minimum phase channel impulse response in z-domain having only two taps $F(z) = f_0 + f_1 z^{-1} = 1 + 0.5z^{-1}$. Let us denote intermediate results at the variable nodes as $\hat{v}_q \in \{+1, -1\}$, where q is the node index. Then, the equalization step is given as

$$y_q - f_1 \cdot \hat{v}_{q-1} \ . \tag{6.1}$$

This equalized value is used to recompute the intrinsic LLR at every iteration according to Eq. (2.37). For a channel having M_t taps, the generalized form of the equalized value is

$$y_q - \sum_{i=1}^{M_t-1} f_i \cdot \hat{v}_{q-i} \ . \tag{6.2}$$

The decoding block shown in Fig.6.5 replaces the serial operations of equalizer and LDPC decoder. For this joint method, two different scheduling methods can be considered.

In the first iteration, the decoding process starts at the LDPC decoder. The received analog values are used directly in the variable to check node iterations, without yet having done an equalization step. In the first iteration in LDPC decoding, the intrinsic LLR values are directly forwarded from the variable nodes to the check nodes, and the check nodes process the information and send it back to the variable nodes. The first equalization step is performed using the values from the variable nodes, after this first iteration. Then similarly, after every iteration in the Tanner graph, equalization is performed.

For the second alternative, we start the decoding process by first equalizing the received values. This first equalization step is similar to a DFE, except it is performed according to the structure in Fig. 6.5. This means that, each value is equalized by using only the previous received value, the recursive processing effect of a DFE, as shown in Fig. 6.2, is lost. The first sum-product algorithm iteration is performed using these equalized values. Thereafter, the iterative process continues according to Eq. (6.1). We compare the two methods with a serial scheme in which equalization is performed according to Fig. 6.2 and a subsequent LDPC decoder uses the equalized values, without any iterative equalization.

6.2.2 Simulation Results

A rate-$1/2$ irregular LDPC code of length $N = 2048$ with the following degree distribution polynomials was simulated:

$$\begin{aligned} \lambda(x) &= 0.28286x + 0.39943x^2 + 0.31771x^7, \\ \rho(x) &= 0.6x^5 + 0.4x^6. \end{aligned}$$

Similar to the previous chapters, the $\mathbf{H}$-matrix was constructed using the triangular zigzag-PEG algorithm described in Chapter 2. All the BER performance curves were obtained after a maximum of 20 iterations.

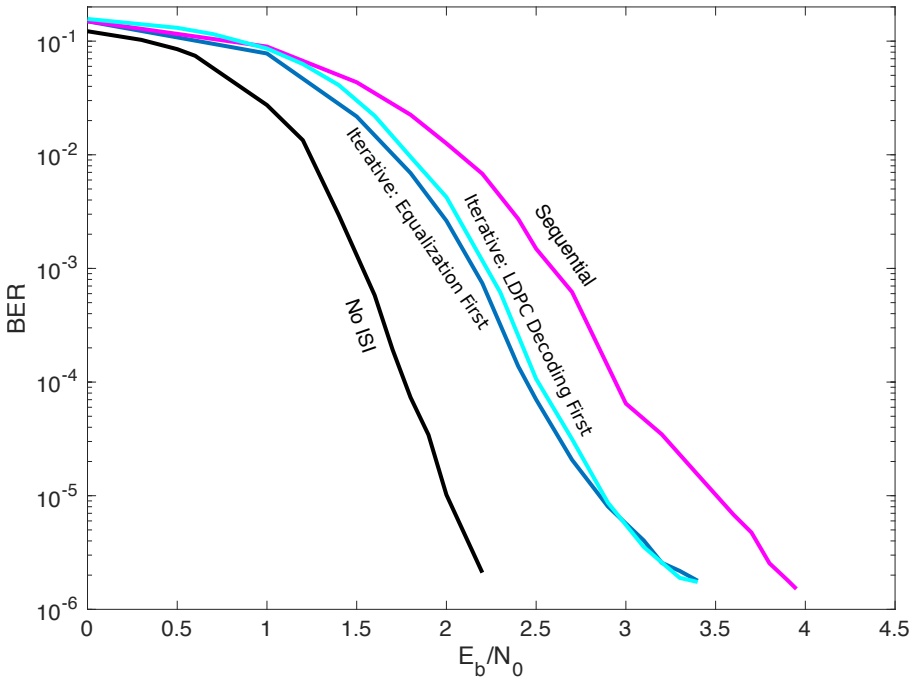


Figure 6.6: Comparing different equalization methods

The results for all three methods are shown in Fig. 6.6. A reference curve is also provided which shows the performance of the code without ISI as a lower bound for the BER. The sequential method, as expected, has the worst performance, since the equalized values are not updated. Of the two variants of the iterative approach discussed previously, we note that there is a slight advantage provided by the method which uses equalized values as a starting point for LDPC decoding. This result is expected due to the fact that when iterations are started inside the LDPC decoder itself, without having performed any equalizing step, the initial intrinsic LLR values are less reliable.

6.2.3 Summary

In this chapter, we have discussed how to perform iterative equalization inside the Tanner graph. We showed a simple approach which uses intermediate LDPC decoding results for iterative equalization. The proposed method is computationally simpler than Turbo equalization. From the results, we see that the proposed methods work as expected, by improving the BER performance with very minor adjustments at the decoder. The methods shown here are proposed as an alternative to serial equalization and decoding, since it delivers gains at very little computational cost.

Chapter 7

Conclusion

In this thesis, we investigated the statistical bindings present in three parts of a transmission system, the source, the channel, and the noise model. The statistical bindings can be characterized via Markov models, for the source and the noise, and channel memory presents itself as ISI. Considering such bindings can lead to modified LDPC decoding algorithms. For each of the three cases, we investigate methods for including the dependencies in the LDPC decoder, in order to use the relations present within a transmission scheme in an efficient manner. A code optimization method is also investigated, with the purpose of determining how dependancies within the Tanner graph are to be handled during the density evolution procedure.

When an information source is modeled by a Markov chain, the resulting decoder should include the dependencies which are present in the low-entropy source. Considering the source model to be known at the receiver, this information can be incorporated via additional edges inside the Tanner graph between neighboring variable nodes, a structure which follows the model at the source. Using intermediate results of the belief-propagation iterations, the variable-node symbol estimates can yield information about the subsequent source symbols. We see this computed information as providing an a-priori estimate of the information, although computed at the decoder iteratively. The resulting performance curves show that there is an advantage to be gained by including such information. The computation for using the source model dependency is simplified here by using Jensen's inequality on the LLR expression which computes the dependency. In simulations, we observe that the simplified computation provides a similar error performance. A Turbo-like scheme is also used in which a BCJR decoder and an LDPC decoder iteratively exchange extrinsic information in order to decode the source. The BCJR decoder estimates the source sequence using the Markov model, using the extrinsic information provided by the LDPC code constraints and vice versa. The Turbo scheme achieves a better performance at low SNRs when compared to the in-Tanner-graph

methods, albeit at higher computational complexity.

Subsequent to the presented decoding methods, we then looked into code structure optimization for the in-Tanner-graph method. While the *left-to-right* links carry a-priori information, the information is computed as a function of the variable node estimates at every iteration, which is computed by adding all the edges that are incoming to an information variable node. Hence, the (a-priori) information carried on such edges is a function of the edge degree polynomials of the information nodes. The information evolution on these links are incorporated into the average mutual information computation equations, which are required for calculating the performance of a code. The optimization procedure requires density evolution as an 'inner' algorithm for determining code performance, with an 'outer' algorithm searches for the code which is the most suitable, based on an appropriate objective function. Here, the rate-$1/2$ code which has the best convergence threshold was chosen as the objective. The density evolution procedure is a mixture of *full density evolution*; i.e., operations on densities, as well as an approximation to the procedure known as the Gaussian approximation to density evolution. From the simulation results, we observe that the BER curves of the optimized codes are steeper than non-optimized codes.

For impulse noise, the Middleton Class-A model was considered. The memory in the process was described by a Markov model. The MCA model approximates the noise amplitude by a Gaussian mixture model. Here we considered two terms for the noise mixture, with one representing the background Gaussian noise state and the other representing an impulsive state. A Markov model is used to map the transitions between impulsive and Gaussian states, which leads to a model describing the inter-arrival time of noise impulses. For decoding, a Turbo-like scheme is proposed, with a Viterbi algorithm decoding the sequence of noise-states and an LDPC decoder decoding the information symbols. We also looked at possibilities for estimating the noise states inside the Tanner graph directly. The additive noise process is not directly related to the code structure and cannot be effectively estimated inside the Tanner graph. In the case of source memory, the source symbols directly map onto the information variable nodes, hence, iterative improvements in the variable node processing can be used to estimate the original symbols inside the Tanner graph, via a function that models the Markov chain dependancies. This analogous treatment is not possible for impulse-noise-with-memory cancellation. A dedicated decoder, like the Viterbi algorithm, is required for noise-state estimation, leading to a Turbo-like scheme as the solution.

The last step consisted of cancelling inter-symbol interference due to channel memory. An iterative scheme is employed where intermediate symbol estimates of the LDPC decoder are

used to equalize the received analog values. Subsequently, the improved received analog values lead to updated values for the intrinsic information available at the LDPC decoder, the two processes iteratively improving the overall correction of the received word.

Both the source and channel memory processes create dependencies between neighboring variable nodes. The distinguishing factor being the order in which memory representation functions appear. For the source, the additional memory is present before LDPC encoding while the ISI channel establishes the memory after encoding. Also, the source memory model is stochastic, while the channel memory model is deterministic. The resulting modification at the decoder for the channel leads to a deterministic correction of the received analog values, which is a modification in the intrinsic information, in an iterative fashion. The source model on the other hand influences the a-priori information following the stochastic Markov source model.

The theoretical exploration of how to handle memory in different parts of the transmission chain are concluded with the steps detailed above. Based on the insights gained from this work, we outline some future areas of investigation. For impulse noise, the considered Middleton Class-A model addresses the inter-arrival time of noise impulses and within a state the active noise model is considered Gaussian. This Gaussian density of the impulse noise can be updated to incorporate practical impulse noise densities which have certain time-dependent behavior. This would lead to additional dependencies between noise samples within the active impulse-noise window, requiring a more detailed treatment of handling impulse-noise-with-memory. For the source model with memory, the low entropy source was not compressed before transmission. A joint source-channel coding scheme which includes syndrome compression of the source, thus leading to a more efficient overall joint code would be the next step for investigation.

List of Symbols

A	duty cycle of impulse noise in MCA model
$\mathcal{C}$	linear code
C	capacity
c	codewords of code $\mathcal{C}$
D	number of samples affected by impulse noise
d_c	maximum check node degree
d_H	Hamming distance
d_v	maximum variable node degree
E	emission probability matrix for a hidden Markov model
E_s	energy per transmitted symbol
E_b	energy per information bit
$\mathbb{F}_q$	finite field having q elements
$F(z)$	minimum phase channel impulse response in z domain
$f(\cdot)$	pdf of the argument of f
Γ	describes the power ratio between background and impulse noise
$\Gamma_{t-1}(s')$	surviving path metric upto state s' at time $t-1$ in a trellis
$\mathbf{G}$	generator matrix of a linear code
G	background Gaussian noise state
g_{mi}	function computing the mutual information x_{vc} from the density of a variable node LLR message and the distribution of the input binary transmitted source
g_t	density transform function
$\mathbf{H}$	parity-check matrix of a linear code
H	entropy
I	impulse noise state
i	information vector of a linear code

J	J-function for computing the mutual information from the mean or variance of a consistent Gaussian density
K	length of an information vector of a linear code
$\lambda_t(s', s)$	branch metric in a trellis between states (s', s)
$\lambda(x)$	variable-node edge degree distribution polynomial
$\tilde{\lambda}(x)$	variable-node degree distribution polynomial
$L(x_i)$	LLR for the i^{th} bit of $\mathbf{x}$
L_A	a-priori LLR value
$L(v_q)$	LLR at the v^{th} variable node
L_{itr}	intrinsic LLR value
$L_{v\to w}$	LLR message from variable node v to check node w
$L_{w\to v}$	LLR message from check node w to variable node v
l	iteration counter during SPA algorithm
μ	mean of a Gaussian distribution
M	length of a parity vector of a linear code
M_t	taps of a channel filter function
m	number of noise states in an impulse noise environment
m_n	noise state active during the n^{th} bit transmission
N	length of a codeword of a linear code
N_0	two-sided noise power spectral density
N_s	number of states in a finite state machine
N_z	noise power
O	random variable describing the output of a hidden Markov model
π	stationary distribution of a Markov model.
P	signal power
p_1	$P(s = 0\|s' = 0)$ for a 2-state Markov chain with states $s_i \in \{0, 1\}$
p_2	$P(s = 1\|s' = 1)$ for a 2-state Markov chain with states $s_i \in \{0, 1\}$
Q	transition-probability matrix of a Markov model
Q_D	transition probability matrix of the MCA model for a value D
Q_s	transition probability matrix of a binary source alphabet
q_1	$P(I\|G)$: transition probability from background noise state to impulse noise state

q_2	$P(G\|I)$: transition probability from impulse noise state to background noise state
$\rho(x)$	check-node edge degree distribution polynomial
$\tilde{\rho}(x)$	check-node degree distribution polynomial
R	rate of a code
σ^2	variance of a Gaussian distribution
S	random variable denoting the states of a Markov chain
s_i	states of a finite state machine describing a Markov model where $i \in \{0, 1, ..., N_s\}$
s'	previous state in a state sequence
s	current state in a state sequence
Θ	parameterized hidden Markov model
t	discrete-time index
T_D	duration of impulse noise
T_e	symbol duration
$\mathbf{u}$	a binary source information sequence
u	a binary source information symbol
v_i	variable node i
$\hat{v}_q$	intermediate BP estimate at variable node v_q
W	bandwidth
w_i	check node i
$\mathbf{x}$	a transmitted vector
x_{cv}	mutual information from check nodes to variable nodes
x_{cv}	mutual information from variable nodes to check nodes
x_q	q^{th} bit of transmit vector $\mathbf{x}$
$\hat{x}_q$	decoder estimate of x_q
$\mathbf{y}$	a received vector
$\tilde{y}$	intermediate estimate for the received bit y
y_q	q^{th} bit of received vector $\mathbf{y}$
$\mathbf{z}$	noise vector

Acronyms

APP	A-Posterior Probability
BP	Belief-Propagation
BPSK	Binary Phase Shift Keying
CN	Check Nodes
CTMC	Continuous Time Markov Chain
CSI	Channel-state Information
DE	Density Evolution
DFE	Decision-Feedback Equalization
DTMC	Discrete Time Markov Chain
FEC	Forward Error Correction
HMM	Hidden Markov Model
ISI	Inter-Symbol interference
LDPC	Low-density Parity-check
LLR	Log-Likelihood Ratio
LMS	Least Mean Squares
MAP	Maximum A-posteriori Probability
MCA	Middleton Class-A
MI	Mutual Information
ML	Maximum-Likelihood
MMSE	Minimum Mean Squared Error
pdf	Probability Density Function
RV	Random Variable
SNR	Signal-to-Noise Ratio
SPA	Sum-Product Algorithm
VN	Variable Nodes
ZF	Zero-forcing

Own Publications

[OWN01] W. Henkel, **N. S. Islam**, and M. A. Leghari, "Joint equalization and LDPC decoding," in *11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2019, pp. 1–5.

[OWN02] **N. S. Islam** and W. Henkel, "Information Forwarding in LDPC Decoding for Markov Sources," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, 2018, pp. 1–5.

[OWN03] W. Henkel, O. A. Graur, **N. S. Islam**, U. Pagel, N. Manak, and O. Can, "Reciprocity for physical layer security with wireless FDD and in wireline communications," in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018, pp. 1–6.

[OWN04] K. A. Saaifan, **N. Islam**, and W. Henkel, "Lattice coding for MIMO systems in impulse noise," in *2017 9th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2017, pp. 220–225.

[OWN05] **N. Islam**, O. Graur, and W. Henkel, "Physical layer key generation using FDD wireless and powerline channels," in *ITG Fachgruppe "Angewandte Informationstheorie"*, Bremen, 2017.

[OWN06] O. Graur, **N. Islam**, A. Filip, and W. Henkel, "Quantization and LLR computation for physical layer security," in *International Zurich Seminar on Communications (IZS 2016)*, Zurich, 2016

[OWN07] O. Graur, **N. Islam**, and W. Henkel, "Quantization for physical layer security," in *2016 IEEE Globecom Workshops (GLOBECOM), Workshop on Trusted Communications with Physical Layer Security*, Washington DC., 2016, pp. 1–7.

[OWN08] R. Mehmood, R. Sharma, J. Wallace, O. Graur, **N. Islam**, A. Filip, and W. Henkel, "Physical-Layer key generation and reconciliation" in *Communications in Interference Limited Networks*, Springer, pp. 393-430, ISBN: 978-3-319-22440-4, 2016.

[OWN09] **N. Islam**, O. Graur, A. Filip, and W. Henkel, "LDPC code design aspects for physical-layer key reconciliation," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–7.

[OWN10] O. Graur, **N. Islam**, A. Filip, and W. Henkel, "Quantization aspects in LDPC key reconciliation for physical layer security," in *SCC 2015; 10th International ITG Conference on Systems, Communications and Coding*, 2015, pp. 1–6.

[OWN11] A. Mahmood, **N. Islam**, D. Nigatu, and W. Henkel, "DNA inspired bi-directional Lempel- Ziv-like compression algorithms," in *2014 8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2014, pp. 162–166.

[OWN12] O. Graur, **N. Islam**, A. Filip, and W. Henkel, "Reconciliation procedures for physical-layer key generation," in *Sitzung der ITG Fachgruppe "Angewandte Informationstheorie"*, Technische Universitaet Muenchen, 2014.

[OWN13] O. Graur, A. Filip, **N. Islam**, and W. Henkel, "Physical layer security based on vector quantization and Slepian-Wolf coding," in *3rd International Workshop on Advances in Communications*, Boppard, Germany, 2014.

List of Figures

Bibliography

[1] A. A. Markov, "An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains," *Science in Context*, vol. 19, no. 4, p. 591–600, 2006.

[2] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[3] D. Jurafsky and J. H. Martin, *Speech and Language Processing, 3rd Ed. Draft*, n.d. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[4] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[5] G. Forney, "The Viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

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

[7] R. G. Gallager, "Low-density parity-check codes," Ph.D. dissertation, MIT, Cambridge, Mass., USA, 1963.

[8] D. J. C. Mackay and R. Neal, "Good codes based on very sparse matrices," ser. 5th IMA Conference on Cryptography and Coding. Lecture Notes in Computer Science number 1025, October 1995, Oct. 1995.

[9] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

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

[11] M. Luby, M. Mitzenmacher, A. Shokrollah, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 249–258. [Online]. Available: https://doi.org/10.1145/276698.276756

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

[13] ——, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[14] W. E. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[15] D. J. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, 2003, mFCSIT 2002, The Second Irish Conference on the Mathematical Foundations of Computer Science and Information Technology. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066104807680

[16] T. Richardson, "Error floors of LDPC codes," 2003.

[17] B. Vasić, S. Chilappagari, D. Nguyen, and S. Planjery, "Trapping set ontology," in *2009 47th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2009*, ser. 2009 47th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2009, Dec. 2009, pp. 1–7, 2009 47th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2009 ; Conference date: 30-09-2009 Through 02-10-2009.

[18] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Department of Electrical Engineering, Linköping University, Sweden, 1996.

[19] B. Frey, R. Koetter, and A. Vardy, "Skewness and pseudocodewords in iterative decoding," in *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No.98CH36252)*, 1998, pp. 148–148.

[20] ——, "Signal-space characterization of iterative decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 766–781, 2001.

[21] P. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," *ArXiv*, vol. abs/cs/0512078, 2005.

[22] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Predicting error floors of structured LDPC codes: deterministic bounds and estimates," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 908–917, 2009.

[23] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1570–1579, 2002.

[24] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, "Stopping sets and the girth of Tanner graphs," in *Proceedings IEEE International Symposium on Information Theory,*, 2002, pp. 2–2.

[25] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Techn. Journal*, vol. 27, pp. 623–656, 1948.

[26] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of Capacity-approaching Irregular Low-density Parity-check Codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, 2001.

[27] Sae-Young Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of Sum-product Decoding of Low-density Parity-Check Codes using a Gaussian Approximation," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 657–670, 2001.

[28] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Transactions on Communications*, vol. 49, no. 10, pp. 1727–1737, 2001.

[29] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 670–678, 2004.

[30] S. ten Brink and G. Kramer, "Design of repeat-accumulate codes for iterative detection and decoding," *IEEE Transactions on Signal Processing*, vol. 51, no. 11, pp. 2764–2772, 2003.

[31] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[32] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth Tanner graphs," in *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, vol. 2, 2001, pp. 995–1001 vol.2.

[33] D. MacKay, S. Wilson, and M. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Transactions on Communications*, vol. 47, no. 10, pp. 1449–1454, 1999.

[34] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 638–656, Feb. 2001.

[35] T. Tian, C. Jones, J. Villasenor, and R. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Transactions on Communications*, vol. 52, no. 8, pp. 1242–1247, 2004.

[36] D. Vukobratovic and V. Senk, "Generalized ACE constrained progressive edge-growth LDPC code design," *IEEE Communications Letters*, vol. 12, no. 1, pp. 32–34, 2008.

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

[38] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[39] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[40] ——, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[41] Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[42] C. Shannon, "Coding theorems for a discrete source with a fidelity criterion," in *Institute of Radio Engineers, International Convention Record*, vol. 7, 1959, pp. 142–163.

[43] T. Cover, A. Gamal, and M. Salehi, "Multiple access channels with arbitrarily correlated sources," *IEEE Transactions on Information Theory*, vol. 26, no. 6, pp. 648–657, 1980.

[44] S. Vembu, S. Verdu, and Y. Steinberg, "The source-channel separation theorem revisited," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 44–54, 1995.

[45] A. Goldsmith, "Joint source/channel coding for wireless channels," in *1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century*, vol. 2, 1995, pp. 614–618 vol.2.

[46] K. Choi, K. Tatwawadi, T. Weissman, and S. Ermon, "NECST: neural joint source-channel coding," *CoRR*, vol. abs/1811.07557, 2018. [Online]. Available: http://arxiv.org/abs/1811.07557

[47] G. Caire, S. Shamai, and S. Verdú, "Almost-noiseless joint source-channel coding-decoding of sources with memory," in *Proc. 5th International ITG Conference on Source and Channel Coding*, Jan. 2004, pp. 295–304.

[48] M. Fresia, F. Pérez-Cruz, and H. V. Poor, "Optimized concatenated LDPC codes for joint source-channel coding," in *Proc. IEEE Int. Symposium on Information Theory*, Seoul, South Korea, 2009.

[49] H. V. Beltrão Neto and W. Henkel, "Multi-edge optimization of low-density parity-check codes for joint source-channel coding," in *Proc. of 9th International ITG conference on Systems, Communications and Coding*, Munich, Germany, Jan. 2013.

[50] ——, "Information shortening for joint source-channel coding schemes based on low-density parity-check codes," in *Proc. of 8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Bremen, Germany, August 2014.

[51] J. He, L. Wang, and P. Chen, "A joint source and channel coding scheme base on simple protograph structured codes," in *2012 International Symposium on Communications and Information Technologies (ISCIT)*, 2012, pp. 65–69.

[52] L. Wang, H. Wu, and S. Hong, "The sensitivity of joint source-channel coding based on double protograph LDPC codes to source statistics," in *2015 9th International Symposium on Medical Information and Communication Technology (ISMICT)*, 2015, pp. 213–217.

[53] H. Wu, L. Wang, S. Hong, and J. He, "Performance of joint source-channel coding based on protograph LDPC codes over Rayleigh fading channels," *IEEE Communications Letters*, vol. 18, no. 4, pp. 652–655, 2014.

[54] J. Garcia-Frias and J. Villasenor, "Joint turbo decoding and estimation of hidden Markov sources," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 9, pp. 1671–1679, 2001.

[55] ——, "Combining hidden markov source models and parallel concatenated codes," *IEEE Communications Letters*, vol. 1, no. 4, pp. 111–113, 1997.

[56] A. Guyader, E. Fabre, C. Guillemot, and M. Robert, "Joint source-channel turbo decoding of entropy-coded sources," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 9, pp. 1680–1696, 2001.

[57] L. Yin, J. Lu, and Y. Wu, "Combined hidden Markov source estimation and low-density parity-check coding: a novel joint source–channel coding scheme for multimedia

communications," *Wireless Communications and Mobile Computing*, vol. 2, no. 6, pp. 643–650, 2002. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcm.86

[58] E. Ordentlich, G. Seroussi, S. Verdu, and K. Viswanathan, "Universal algorithms for channel decoding of uncompressed sources," *IEEE Transactions on Information Theory*, vol. 54, no. 5, pp. 2243–2262, 2008.

[59] J. Hagenauer, "Source-controlled channel decoding," *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2449–2457, 1995.

[60] Z. Mei and L. Wu, "LDPC Codes for Binary Markov Sources." [Online]. Available: en.paper.edu.cn/en_releasepaper/content/794.

[61] N. S. Islam and W. Henkel, "Information Forwarding in LDPC Decoding for Markov Sources," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, 2018, pp. 1–5.

[62] C. Chen, L. Wang, and F. C. M. Lau, "Joint optimization of protograph LDPC code pair for joint source and channel coding," *IEEE Transactions on Communications*, vol. 66, no. 8, pp. 3255–3267, 2018.

[63] G.-C. Zhu and F. Alajaji, "Joint source-channel turbo coding for binary Markov sources," *IEEE Transactions on Wireless Communications*, vol. 5, no. 5, pp. 1065–1075, 2006.

[64] D. Middleton, "Statistical-physical models of electromagnetic interference," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-19, no. 3, pp. 106–127, 1977.

[65] T. Shongwey, A. J. H. Vinck, and H. C. Ferreira, "On impulse noise and its models," in *18th IEEE International Symposium on Power Line Communications and Its Applications*, 2014, pp. 12–17.

[66] H. Nakagawa, D. Umehara, S. Denno, and Y. Morihiro, "A decoding for low density parity check codes over impulsive noise channels," in *International Symposium on Power Line Communications and Its Applications, 2005.*, 2005, pp. 85–89.

[67] M. S. Alam, B. Selim, G. Kaddoum, and B. L. Agba, "Mitigation techniques for impulsive noise with memory modeled by a two state Markov-Gaussian process," *IEEE Systems Journal*, vol. 14, no. 3, pp. 4079–4088, 2020.

[68] M. Tuchler, R. Koetter, and A. Singer, "Turbo equalization: principles and new results," *IEEE Transactions on Communications*, vol. 50, no. 5, pp. 754–767, 2002.

[69] W. Henkel, N. S. Islam, and M. A. Leghari, "Joint equalization and LDPC decoding," in *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2019, pp. 1–5.