UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Marten Moro

Optimized signaling for ultra-high-speed fiber-optical communications

Bachelor's Thesis (9 ECTS)

Supervisor: Vitaly Skachek, PhD
Supervisor: Irina Bocharova, PhD

Tartu 2021

Optimized signaling for ultra-high-speed fiber-optical communications

Abstract: This thesis deals with the topic of channel coding. We start from the basics, defining a channel, a decoder and other channel coding elements gradually moving to more complex topics ending with non-binary low-density parity-check codes. The aim of the thesis is to give the reader a grasp on why do we need the channel coding and how does it work. Finally we implement a fast Hadamard transform and a regular Hadamard transform functions in Matlab and compare their performances when used for decoding non-binary low-density parity-check codes.

Keywords:
Channel coding, error-correction code, low-density parity-check code

CERCS: P170 Computer science, numerical analysis, systems, control

Ülikiirel valgustehnoloogial põhinev optimeeritud suhtlus

Lühikokkuvõte: Käesolev töö tutvustab lugejale, mis on kanalikodeerimine. Alustatakse baasteadmistest, määrates sealjuures ka ära, mis on kanal, mis on dekooder ning teised elemendid. Järk-järgult liigutakse keerulisemate ning spetsiifilisemate teemade peale. Töö eesmärk on anda lugejale aimdus, mis on kanalikodeerimine, miks me seda vajame ning kuidas see töötab. Lõpetuseks implementeeritakse ka kiire Hadamardi teisendus ning tavaline Hadamardi teisendus Matlabis ning võrreldakse nende kiirust mittebinaarsete hõredate paarsuskontrolli koodide dekodeerimisel.

Võtmesõnad:
Kanalikodeerimine, veaparanduskood, hõre paarsuskontrolli kood

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Nowadays, ever increasing amounts of data are transmitted and stored. To fulfill the demand for higher data rates we need to keep enabling the next generation technologies, like fiber-optical transport networks. Low-density parity-check (LDPC) codes, which we will be looking in this thesis are widely used for error correction in such networks [7].

A channel is a medium between the source and the destination, which is used for data transmission. A good example of the source and the destination can be telephone sets, and the channel between them would be the telephone line. Today most of communications are in the digital form, which means that the information is represented by binary symbols. The channel itself is still analogous, but the receiver converts the analogous signal into discrete-time values. We can model this channel with additive white Gaussian noise. One example of a digital channel would be a fiber-optical cable. Every channel can have disturbances in the data transmission process, and we protect the information from these disturbances by using channel coding. For example, if we send a binary vector through the channel and the noise is so significant that we receive a vector with some flipped bits, then channel coding helps us to correct the errors. To do so, we add redundancy to the information that is being sent, the channel encoder then encodes the information to have a certain structure and at the endpoint, the channel decoder, while decoding the message, uses the redundancy to correct errors that happened in the transmission process. At the moment, the main bottleneck in the data transmission is the decoding process [9].

Every channel has its capacity, known as Shannon limit, introduced by Claude Shannon in 1948. The Shannon limit describes the maximum rate at which data can be transmitted over the channel error-free given its bandwidth and noise characteristics [10]. We have found codes that are performing near Shannon limit, but the one that performs on Shannon limit is yet to be found.

The goal of this thesis is to introduce the reader to the topic of channel coding. Even though there is a variety of error-correcting codes and corresponding decoding algorithms, in this thesis we will end with non-binary (NB) low-density parity-check codes.

We first start with introducing the channel coding, channel models and error-correcting codes. Moving forward we look at the maximum a posteriori (MAP) decoding and MAP decoding over trellises using BCJR algorithm. This is also used in decoding LDPC codes. When looking at the LDPC codes, we first look at the hard-decision decoding method, this helps the reader to understand the structure of decoding LDPC codes, but is not really used in the real world, because the soft-decision decoding yields much better results. Then we look at the soft-decision decoding and finally we get to the non-binary LDPC codes using Hadamard transform based decoding. We also implement a fast Hadamard transform and a regular Hadamard transform functions in Matlab and compare their performances.

2 Channel coding

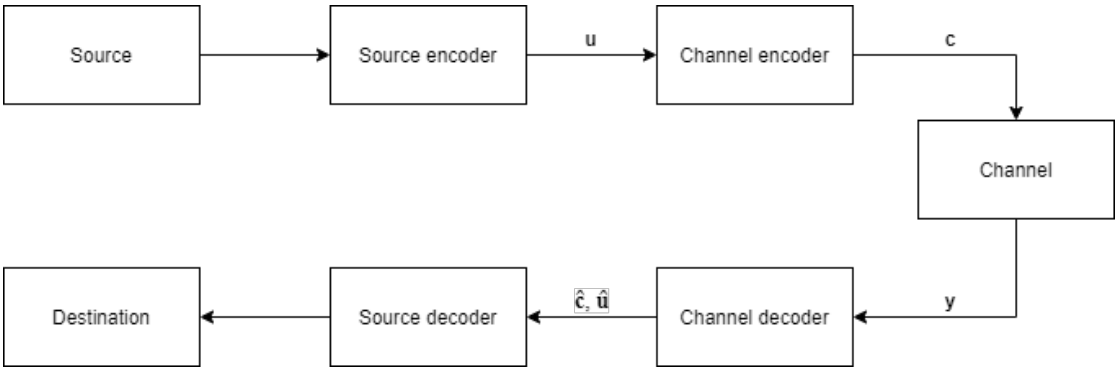

Figure 1. Communication system model

Channel coding consists of three components: channel encoder, channel and the channel decoder. The input to the channel encoder is called information sequence $\mathbf{u}$, which channel encoder encodes to the codeword $\boldsymbol{c}$. The codeword is the actual data vector that will be sent through the channel. From transmission over channel we receive the vector $\boldsymbol{y}$. Usually $\mathbf{y} \neq \mathbf{c}$ due to the noise in the channel. This means that some bits of $\boldsymbol{c}$ may have flipped during the transmission process or in case of an additive white Gaussian noise channel, a signal with a flipped sign. We define the channel S as a triple (F, ϕ, Prob), where F is a finite input alphabet, ϕ is a finite output alphabet and Prob is a conditional probability distribution

$$\texttt{Prob}\left\{\mathbf{y}|\mathbf{c}\right\}, \tag{1}$$

where $\boldsymbol{c}$ is the transmitted codeword over F^n and $\boldsymbol{y}$ is the received vector over ϕ^n. After transmission, the channel decoder corrects the errors in the received information. In case of a perfect decoding process $\mathbf{c} = \hat{\mathbf{c}}$ and $\mathbf{u} = \hat{\mathbf{u}}$ [13].

3 Channel models

3.1 Memoryless binary symmetric channel

Memoryless binary symmetric channels (BSC) are channels, where $F = \phi = \{0, 1\}$ and the channel being memoryless means that the channel noise for every symbol is independent from the noise affecting previous symbols. For every codeword word $\boldsymbol{c} = (c_1, c_2, ...c_n)$ and received word $\boldsymbol{y} = (y_1, y_2, ...y_n)$ with the length of n

$$Prob\{\boldsymbol{y}|\boldsymbol{c}\} = \prod_{j=1}^{n} Prob\{y_j|c_j\}, \tag{2}$$

where, for every $c_j, y_j \in F$,

$$Prob\{y_j|c_j\} = \begin{cases} 1-p \ \texttt{if} \ \ y_j = c_j \\ \quad p \ \texttt{if} \ \ y_j \neq c_j, \end{cases}$$

where $0 \leq p \leq 1$. The parameter p is called crossover probability of the channel and it is the probability for the bit to flip when being transmitted over the channel [13].

3.2 Additive white Gaussian noise channel

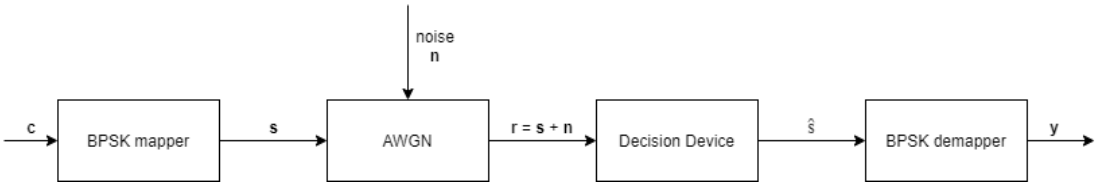

Figure 2. BPSK model

In this thesis we will be dealing with additive white Gaussian noise (AWGN) channel, which is more complex than the BSC. There are different types of modulation techniques, which could be used with the AWGN channels, but we are looking at an AWGN channel with a binary phase shift keying (BPSK) modulation.
The main idea behind the BPSK is that the codeword $\boldsymbol{c}$ is transformed into signals $\boldsymbol{s}$, where 1 and 0 are mapped to certain symbols, for example 0 is mapped to -1 and 1 is mapped to +1. The signals are then sent through the channel, where the noise $\boldsymbol{n}$ is added to the signal [15]. The noise can come from many different natural sources and is independent for every bit [14]. The decision device receives a signal vector $\boldsymbol{r}$, so that $\boldsymbol{r} = \boldsymbol{s} + \boldsymbol{n}$, where n_i is a noise sample. Noise samples are independent Gaussian variables with zero mean and variance $\frac{N_0}{2}$. This channel is characterized by signal-to-noise ratio $\mathsf{SNR} = \frac{E}{N_0}$, where E is a signal energy. So-called hard decision decoding of bits can be done by comparing signal r_i to 0. The decision is 1 if $r_i > 0$ and 0 otherwise. When using soft decision decoding, then signal symbols are mapped to probabilities. Though this way of transmitting signals leaves us some space for noise, sometimes the noise can still be so high, that we receive an erroneous bit [15].

4 Block codes

A block code C is a set of vectors, which we call codewords. A Hamming distance $d(\boldsymbol{x}, \boldsymbol{y})$ is the number of positions where two codewords differ from each other. A minimum distance of C is the minimum Hamming distance between any two distinct codewords of C. The Hamming weight w of a codeword is a number of nonzero entries in it. We say that $C \subset F^n$ is an (n, M, d) code over a finite alphabet F if C has a length n, it contains M codewords, and its minimum distance is d. The bigger the minimum distance, the more distinguishable are different codewords, so it is more likely to get the correct decoded word from the received vector, but it also makes the M smaller. The dimension k of C is $k = \log_{|F|} M$, which shows the length of an actual information that is being sent. The rate R is $R = k/n$, which shows the code efficiency. So for example if $R = 2/3$, then two bits of information are transmitted by sending three bits.
For example if we have a (3, 4, 2) code $C = \{000, 011, 101, 110\}$ over $F = \{0, 1\}$, then $k = \log_2 4 = 2$ and $R = 2/3$. Minimum distance d of C is 2 is because the minimum Hamming distance between any two distinct codewords of C is at least two. For example if $\boldsymbol{c}_1 = (0, 1, 1)$ and $\boldsymbol{c}_2 = (1, 0, 1)$, then they are differing in the first and the second position [13].

5 Linear codes

Linear block codes are codes, with a certain structure. Linear codes are determined by a generator matrix and a parity-check matrix, which are used for encoding and decoding the code. The alphabets for linear codes are finite fields or so called Galois fields. It means that $F = GF(q)$, where q is the size of the field. The code (n, M, d) is a linear code if it meets these two requirements:

$$1)\ \ \boldsymbol{c}_1, \boldsymbol{c}_2 \in C \Rightarrow \boldsymbol{c}_1 + \boldsymbol{c}_2 \in C \tag{3}$$

$$2)\ \ \boldsymbol{c} \in C \ \text{ and } \ \alpha \in F \Rightarrow \alpha \boldsymbol{c} \in C\,, \tag{4}$$

where α is a scalar [13].

5.1 Generator matrix

If block codes are usually written as (n, M, d) codes, then linear codes are usually written as [n, k, d] codes, because we use only k rows in the generator matrix to produce M codewords. The rows of the generator matrix are the basis of the code, for example if we have a (3, 4, 2) single parity check code over $GF(2)$ {(0,0,0), (0,1,1), (1,0,1), (1,1,0)}, then its generator matrix

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

because by using a generator matrix codewords can be obtained as linear combinations of its rows. In the current example we can do the following equations to get the two codewords that are not listed in the generator matrix:

$$0\mathbf{c}_1 = (0, 0, 0) \tag{5}$$

$$\mathbf{c}_1 + \mathbf{c}_2 = (1, 1, 0). \tag{6}$$

Encoding information word $\boldsymbol{u} \in F^k$ to codeword $\boldsymbol{c}$ is performed as

$$\boldsymbol{c} = \boldsymbol{u}G \tag{7}$$

A $k \times n$ generator matrix is called systematic if it has the form

$$G = \begin{pmatrix} I & | & A \end{pmatrix},$$

where I is a $k \times k$ identity matrix and A is a $k \times (n - k)$ matrix [13]. In our example

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

and

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

5.2 Parity-check matrix

Parity-check matrix H can be used for decoding of linear codes. Parity-check matrix satisfies

$$HG^T = 0 \Rightarrow GH^T = 0, \tag{8}$$

and for every vector $\mathbf{c} \in F^n$,

$$\boldsymbol{c} \in C \iff H\boldsymbol{c}^T = \mathbf{0} \tag{9}$$

where $(\cdot)^T$ stands for transposition. The syndrome of the received word $\boldsymbol{y}$ is defined by

$$\boldsymbol{s}^T = H\boldsymbol{y}^T. \tag{10}$$

This means, that if $\boldsymbol{s} = 0$, then $\boldsymbol{y}$ is a codeword [13].
Parity-check matrix can be found by finding a basis of the kernel of a generator matrix, or when G is systematic, then $H = (-A^T|I)$ [13].

6 Decoding

A decoder can be represented by a function

$$D : \phi^n \rightarrow C \, , \tag{11}$$

where ϕ^n is a finite output alphabet. The decoding error probability P_{err} of D is defined by

$$P_{err} = \mathtt{max} \;\; P_{err}(\boldsymbol{c}), \tag{12}$$

where

$$\boldsymbol{c} \in C \tag{13}$$

and

$$P_{err}(\boldsymbol{c}) = \sum_{y:D(\boldsymbol{y}) \neq \boldsymbol{c}} Prob\left\{\boldsymbol{y}|\boldsymbol{c}\right\}. \tag{14}$$

P_{err} is a probability that the codeword will be decoded erroneously.
To resolve errors, that may have happened in the transmission process, we use redundancy. For example lets use a binary (3,2,3) repetition code. This is the most basic error-correcting code, that just repeats the message n times. As it can be seen from Table 1, if we want to send a message 0, we will send three zeros. The decoder makes its decision based on the "Decision Region" in Table 1. If the received word belongs to one of the decision region values corresponding to zero, the decoder decodes it as 0. The behaviour of $u = 1$ is analogical. This way of adding redundancy would help to correct all single bit errors, but we are looking for an error correction system which would need the least amount of redundancy, but has the best probability for error correction [13]. This is a codeword optimal method, which means that the decisions will be made for codewords.

Message	**Codeword**	**Decision Region**
0	000	000, 001, 010, 100
1	111	111, 110, 011, 101

Table 1. Simple example of error correcting codes.

6.1 Maximum a posteriori decoding

The purpose of maximum a posteriori (MAP) decoding is to give optimal decision about transmitted symbols, which makes it a symbol optimal algorithm. A posteriori probabilities of symbols $\hat{c} \in \{0, 1\}$ in position t are

$$p(\hat{c}_t = \hat{c}_t|\boldsymbol{y}) = \frac{p(\hat{c}_t = \hat{c}_t, \boldsymbol{y})}{p(\boldsymbol{y})}, \tag{15}$$

where

$$p(\hat{c}_t = \hat{c}_t, \boldsymbol{y}) = \sum_{\boldsymbol{c} \in C_t(\hat{c}_t)} p(\boldsymbol{c}, \boldsymbol{y}), \tag{16}$$

where $C_t(\hat{c})$ stands for a set of codewords with symbol $\hat{c}$ in position *t*,

$$p(\boldsymbol{c}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{c})p(\boldsymbol{c}). \tag{17}$$

Lets consider a BSC with the crossover probability p_0, $\boldsymbol{c}$ is a codeword and $\boldsymbol{c}_m \in C$, and $\boldsymbol{y}$ is the received word, then a posteriori probabilities of codewords are

$$p(\boldsymbol{c}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{c})p(\boldsymbol{c})}{p(\boldsymbol{y})}, \tag{18}$$

where

$$p(\boldsymbol{y}|\boldsymbol{c}) = p_0^e \cdot p_0^{1-e}, \tag{19}$$

where *e* is the amount of non-matching bits in $\boldsymbol{c}$ and $\boldsymbol{y}$,

$$p(\boldsymbol{c}) = \frac{1}{|M|}, \tag{20}$$

$$p(\boldsymbol{y}) = \sum_{m=0}^{|M|-1} p(\boldsymbol{y}|\boldsymbol{c}_m)p(\boldsymbol{c}_m) \tag{21}$$

[1]. For example lets consider a code determined by a generator matrix

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

let $p_0 = 0.1$ and assume that the received word $\boldsymbol{y} = (0, 0, 1, 1, 0)$. Using formulas (15) - (21) we find a posteriori probabilities of codewords ($p(\mathbf{c}|\mathbf{y})$), that are given in Table 2, and a posteriori probabilities of symbols on positions *t*, that are given in Table 3.

u_1, u_2	$\mathbf{c} = (c_0, ..., c_4)$	$p(\mathbf{y}\|\mathbf{c})$	$p(\mathbf{c}\|\mathbf{y})$
00	00000	0.00729	0.296
01	11011	0.00009	0.004
10	11100	0.00081	0.033
11	00111	0.0164025	0.667

Table 2. A posteriori probabilities of codewords.

symbol	$p(\hat{c}_0)$	$p(\hat{c}_1)$	$p(\hat{c}_2)$	$p(\hat{c}_3)$	$p(\hat{c}_4)$
0	0.96	0.96	0.3	0.33	0.33
1	0.04	0.04	0.7	0.67	0.67

Table 3. A posteriori probabilities of symbols on position *t*.

6.2 MAP decoding by using trellis representation

Trellis graph is a labeled directed graph, which states and edges are divided into layers and it is only possible to move from one layer to the next one. If V is a set of states and E is a set of edges in the trellis graph, then we denote layer t of V and E as $V^{(t)} \in V$ and $E^{(t)} \in E$. Paths in the trellis graph are associated with codewords. Figure 3 gives an example of a trellis graph for the code that we used for calculating a posteriori probabilities in Table 2, where *t* is the number of the layer and *s* is the number of the state in the layer [13].

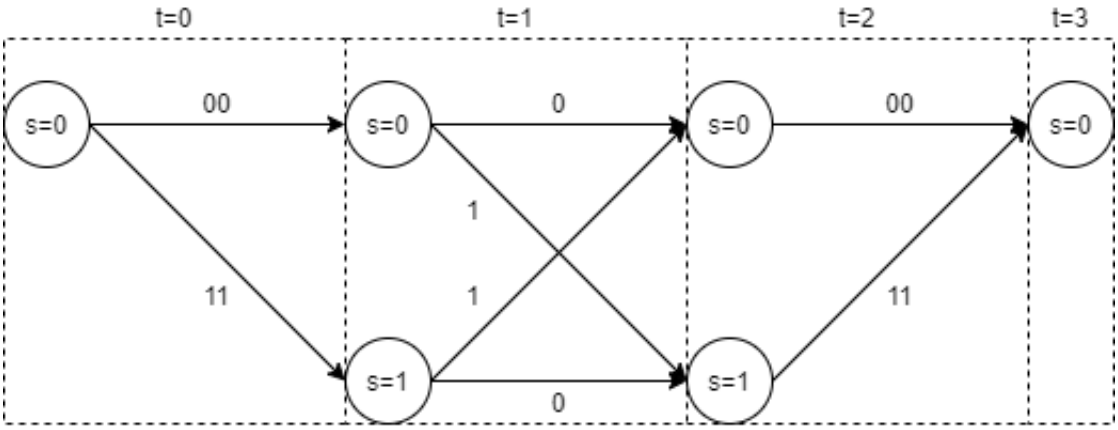

Figure 3. Trellis graph for code used in Table 2

Maximum a posteriori algorithm requires a search over all codewords, which makes its complexity $O(2^k)$, where k is the number of codewords. We want to simplify the maximum a posteriori calculations by using the trellis graph to reduce its complexity to $O(2^u)$, where u is the number of states in the code trellis. This can be achieved by using the BCJR decoding. Let s_t be a state at the layer *t* and $\mathbf{y}_a^b$ a vector of symbols moving from layer *a* to layer *b*, then we can find the conditional probabilities by

$$P(s_{t-1} = m', s_t = m|\mathbf{y}) = \frac{P(s_{t-1} = m', s_t = m, \mathbf{y})}{P(\mathbf{y})}. \tag{22}$$

Lets denote $\sigma_t(m', m)$ as

$$\sigma_t(m', m) = P(s_{t-1} = m', s_t = m, \mathbf{y}) \tag{23}$$

$$\sigma_t(m', m) = P(s_{t-1} = m', \mathbf{y}_1^{t-1}) P(s_t = m, \mathbf{y}_t) P(\mathbf{y}_{t+1}^n) \tag{24}$$

$$\alpha_t(m) = P(s_t = m, \mathbf{y}_1^t) \tag{25}$$

$$\gamma_t(m', m) = P(s_t = m, \mathbf{y}_t | s_{t-1} = m') \tag{26}$$

$$\beta_t(m) = P(\mathbf{y}_{t+1}^n | s_t = m) \tag{27}$$

$$\sigma_t(m', m) = \alpha_{t-1}(m') \gamma_t(m', m) \beta_t(m), \tag{28}$$

where $\alpha_t(m)$ is past, $\gamma_t(m', m)$ is present and $\beta_t(m)$ is future. The computations of $\alpha_t(m)$ and $\beta_t(m)$ are recursions:

$$\alpha_t(m) = \sum_{m'} \alpha_{t-1}(m') \gamma_t(m', m) \tag{29}$$

with initial conditions

$$\alpha_0(m) = \begin{cases} 1, m = 0 \\ 0, m \neq 0 \end{cases}$$

and

$$\beta_t(m) = \sum_{m'} \beta_{t+1}(m') \gamma_{t+1}(m', m) \tag{30}$$

with initial conditions

$$\beta_n(m) = \begin{cases} 1, m = 0 \\ 0, m \neq 0, \end{cases}$$

where

$$\gamma_t(m', m) = p(u_t) p(\mathbf{y}_t | \mathbf{c}_t). \tag{31}$$

Let z be a number of edges from state u_t, then

$$p(u_t) = \frac{1}{z}. \tag{32}$$

$p(\mathbf{y}_t | \mathbf{c}_t)$ is analogous to (19) [2].
Figures 4, 5, 6 and 7 give an example on how to calculate σ.

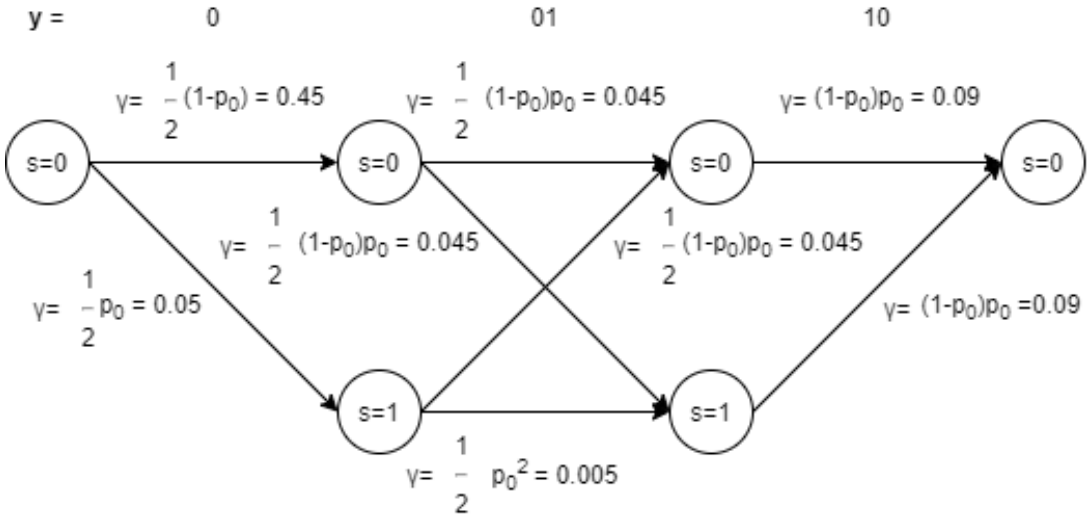

Figure 4. Calculating γ

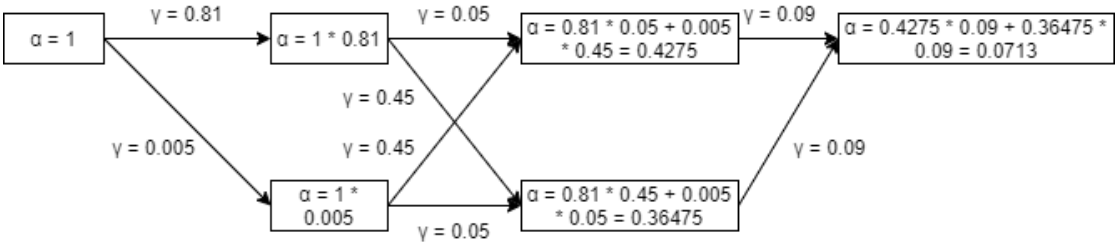

Figure 5. Calculating α

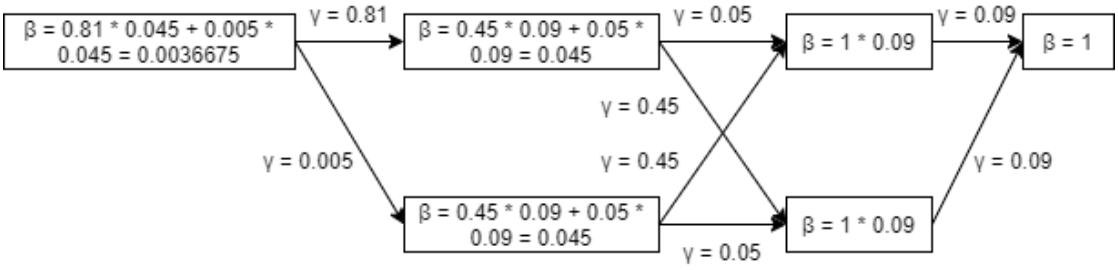

Figure 6. Calculating β

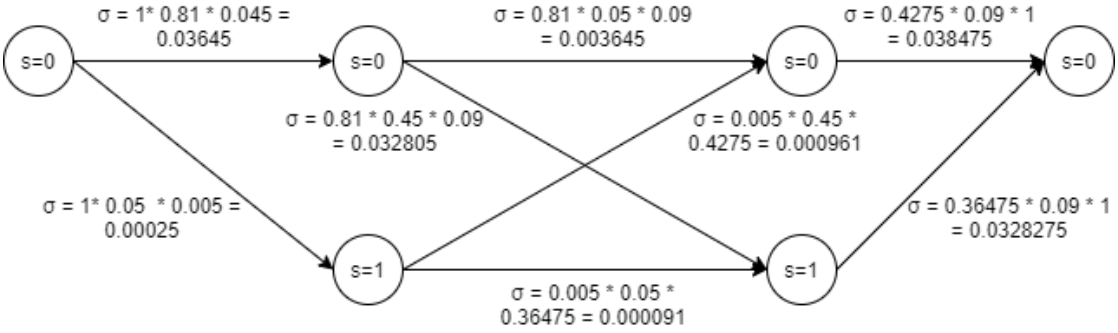

Figure 7. Calculating σ

It is easy to see that the precision required is growing with code length. This problem can be solved by performing calculations in the logarithmic domain. We compute the logarithmic ratios with

$$l = \ln \frac{p(1|y)}{p(0|y)} \tag{33}$$

[12]. For our example calculations of l are shown in Table 4. We can now make a decision using a threshold, so the decision is 1 if $l > 0$ and 0 if $l \leq 0$.

symbol	$\ln \frac{p(1\|y)}{p(0\|y)}$
c_1, c_2	$\ln \frac{0.00025}{0.003645} = -2.68$
c_3	$\ln \frac{0.000961+0.032805}{0.000091+0.003645} = 2.201$
c_4, c_5	$\ln \frac{0.0328275}{0.038475} = -0.159$

Table 4. Logarithms of ratio of a posteriori probabilities

7 Low-density parity-check codes

Low-density parity-check (LDPC) codes are linear codes, which were first introduced in 1963 by Gallager, but were mostly ignored until mid-1990's [3]. In 1981 Tanner generalized LDPC codes and introduced Tanner graphs for a graphical representation of LDPC codes [17].

The codes are called low-density, because the parity-check matrix contains only a few 1's in relation to the amount of 0's. Lets say that we have $n \times m$ matrix, w_r is the weight of rows and w_c is the weight of columns, then the matrix can be called low-density if $w_c \ll n$ and $w_r \ll m$. The LDPC code is considered regular if w_c is a constant and for every row $w_r = w_c(n/m)$ [3].

LDPC codes are represented by their parity-check matrix or by a Tanner graph, which is a graphical representation of the parity-check matrix. This means that construction of LDPC codes usually starts from first constructing the parity-check matrix. LDPC codes are typically tens of thousands bits long, but lets use a LDPC code represented by 4×8 matrix

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

We can see that for every column $w_c = 2$ and for every row $w_r = 4$, which means that this is a regular LDPC code. Tanner graph corresponding to *H* is represented on Figure 8 [3], [17].

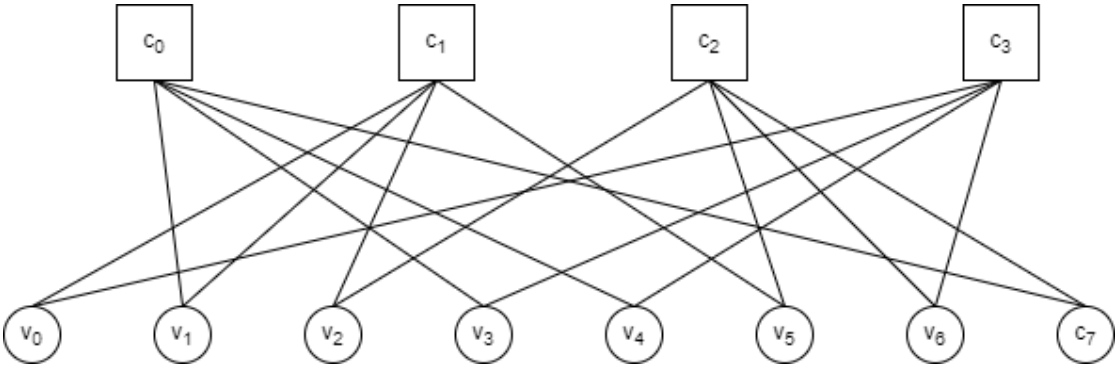

Figure 8. Tanner graph corresponding to *H*

Nodes c_i of the Tanner graph are called check nodes, or c_nodes and nodes v_i are called variable nodes or v_nodes. Variable nodes represent the bits of the received codeword $\boldsymbol{y}$ and check nodes represent check equations. We can see that every bit v_i takes part in two separate check equations (column weight w_c for column i). The LDPC code is regular if

each check node has the same amount of edge connections and if each variable node has the same amount of edge connections [17].
A cycle of length *l* in a Tanner graph is a path of different edges that start and end in the same v_node. In our example there is a length-4 cycle $v_5 \rightarrow c_2 \rightarrow v_2 \rightarrow c_1 \rightarrow v_5$, which is also the shortest cycle length that can be in a Tanner graph. We are interested in cycles, because they degrade the iterative decoding performance, especially the short cycles [17].

7.1 Hard-decision decoding

Hard-decision decoding of LDPC codes is a good way to explain how the decoding of the LDPC codes works, but this is not a method that is actually used for decoding LDPC codes as its preformance is inferior to that of the soft-decision decoding [3].
Let $\mathbf{y} = (0, 0, 1, 1, 1, 0, 0, 0)$ be a received word with an erroneous bit v_3 and $\boldsymbol{c}$ an actually sent codeword $\mathbf{c} = (0, 0, 0, 1, 1, 0, 0, 0)$. The decoding process starts from the decoder receiving $\boldsymbol{y}$. The bits of $\boldsymbol{y}$ are variable nodes v_i.
The first step is that v_nodes (v_i) send the check nodes (c_i) the bit that they believe is the correct one. In the very first iteration it would be the originally received bit. For example in the first iteration c_1 would receive bits $(0, 0, 1, 0)$ [3].
In the second step c_nodes calculate a new bit to every v_node depending on other received bits. This is done separately for every bit and new calculations are not taken into account [3]. The received word $\boldsymbol{y}$ is considered decoded if the syndrome $\boldsymbol{s} = \boldsymbol{y}H^T$ is $\mathbf{0}$. This also means that every parity-check operation has to have even number of ones.
A parity-check equation for one v_node is to check how many ones have been sent by other v_nodes and if the amount is even, the new bit would be 0 and 1 otherwise [17]. In our example if c_1 received bits $(0, 0, 1, 0)$, then the responses from c_1 would be $1 \rightarrow v_0$, $1 \rightarrow v_1$, $0 \rightarrow v_2$, $1 \rightarrow v_5$. The full results from the check equations in the first iteration can be seen in Table 5.
If every parity-check equation is satisfied, this means that none of the bits in any c_node gets changed, the algorithm terminates [3].

c_node	received/sent messages
c_0	received: $v_1 \to 0$, $v_3 \to 1$, $v_4 \to 1$, $v_7 \to 0$ sent: $0 \to v_1$, $1 \to v_3$, $1 \to v_4$, $0 \to v_7$
c_1	received: $v_0 \to 0$, $v_1 \to 0$, $v_2 \to 1$, $v_5 \to 0$ sent: $1 \to v_0$, $1 \to v_1$, $0 \to v_2$, $1 \to v_5$
c_2	received: $v_2 \to 0$, $v_5 \to 0$, $v_6 \to 0$, $v_7 \to 0$ sent: $0 \to v_2$, $0 \to v_5$, $0 \to v_6$, $0 \to v_7$
c_3	received: $v_0 \to 0$, $v_3 \to 1$, $v_4 \to 1$, $v_6 \to 0$ sent: $0 \to v_0$, $1 \to v_3$, $1 \to v_4$, $0 \to v_6$

Table 5. Messages received and sent from c_nodes

In the third step the v_nodes have received new bits from the c_nodes and have to decide which bit they will be using. In the hard-decision algorithm we will just use a majority vote. In this stage the v_node will have received three bits: one is the originally received bit and two are from the c_nodes [3]. For example v_4 has received 0 from the original message, 1 from c_0 and 1 from c_3, the majority is 1 so this will be the new bit for v_4 and it will act as the bit from the original message in the next iteration. The decisions for the v_nodes in the first iteration can be seen in Table 6.

v_node	received bits	decision
v_0	0, 1, 0	0
v_1	0, 0, 1	0
v_2	0, 0, 0	0
v_3	1, 1, 1	1
v_4	1, 1, 1	1
v_5	0, 1, 0	0
v_6	0, 0, 0	0
v_7	0, 0, 0	0

Table 6. Decisions for v_nodes

Next, we proceed with step 2 again. The loop will last as long as the codeword is not found and it is terminated in the second step or if the loop will reach a maximum amount of iterations [3]. In our case the second iteration would terminate the decoding process.

7.2 Soft-decision decoding

The idea behind soft-decision decoding is that the v_nodes will not be sending c_nodes the bits themselves, but instead the probabilities of these bits being in the place they are [3].
Lets introduce some notations:

- $P_i = P(v_i = 1|y)$
- q_{ij} is a message sent from v_node to c_node. This time the message contains a belief that y_i is 1 $(q_{ij}(1))$ and accordingly $1 - q_{ij}(1)$ is a probability, that y_i is 0 $(q_{ij}(0))$.
- r_{ji} is a message sent from c_node to v_node and analogically to q_{ij} we have $r_{ji}(0)$ and $r_{ji}(1)$

In the first step, instead of just sending a bit, we will now also have to calculate q_{ij} and because we only have the single bit at the moment, then $q_{ij} = P_i$ [3]. For calculating P_i we could use the MAP algorithm. Taking into account that each row of the parity-check matrix determines a single parity-check code, the implementation of the MAP algorithm is significantly simplified and can be performed as explained below.
In the second step, for every variable node v_i in check, we now need to calculate the probability that there is an even number of ones in the checking variable nodes except the v_i itself. As we can see from the hard-decision decoding, this is equal to $r_{ji}(0)$, because if there is an uneven amount of ones in the check, we would need to balance it by changing v_i to 1. We can calculate $r_{ji}(0)$ with simplified MAP decoding

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in V_j \setminus i} (1 - 2q_{i'j}(1)), \tag{34}$$

where $V_j \setminus i$ stands for v_nodes in the current check, except v_i [3].
In the third step we calculate new responses for c_nodes and new estimations for v_nodes. To calculate the new q_{ij} we use

$$q_{ij}(0) = K_{ij}(1 - P_i) \prod_{j' \in C_i \setminus j} r_{j'i}(0) \tag{35}$$

$$q_{ij}(1) = K_{ij} P_i \prod_{j' \in C_i \setminus j} r_{j'i}(1), \tag{36}$$

where $C_i \setminus j$ is analogical to (33), but this time we will use all the responses from the c_nodes except the c_j we are currently looking at. The constant K_{ij} is chosen so that $q_{ij}(0) + q_{ij}(1) = 1$ [3].
To calculate probabilities for new estimations to v_nodes v_i we use

$$Qi(0) = K_i(1 - P_i) \prod_{j \in C_i} r_{ji}(0) \tag{37}$$

$$Qi(1) = K_i P_i \prod_{j \in C_i} r_{ji}(1). \tag{38}$$

Note, that this time we use messages from all the c_nodes. Constant K_i is analogical to previously used K_{ij}. To get estimations from probabilities Q_i we can do

$$v_i = \begin{cases} 1 \text{ if } Q_i(1) > Q_i(0), \\ 0 \text{ otherwise} \end{cases}$$

If the new estimated codeword now fulfills the parity-check equations, the algorithm terminates. Otherwise like hard-decision algorithm, it will move to the second step or terminate after the maximum number of iterations [3].

8 Non-binary low-density parity-check codes

The most promising classes of codes for communicating over fiber-optic channels are non-binary low-density parity-check (NB LDPC) codes. Transmission of NB LDPC codes is still done on the binary level, but instead of comparing single bits in the check nodes, we now compare bit vectors.

8.1 Structure

NB LDPC codes are designed over Galois field of characteristic 2 (GF(2^p)). The encoding of NB LDPC codes is similar to encoding binary LDPC codes so that in both cases we get the codeword $\boldsymbol{c}$ from message $\boldsymbol{u}$ with $\mathbf{c} = \mathbf{u}G$, but in case of the NB LDPC codes we have to convert that codeword to a binary form [4].
Let $\boldsymbol{w}$ be a non-binary codeword of the NB LDPC code and $\boldsymbol{c}$ be its binary image. Every element in the field GF(2^p) has a corresponding binary and decimal values. To convert $\boldsymbol{w}$ to $\boldsymbol{c}$ we have to replace decimal values in $\boldsymbol{w}$ with their binary counterparts. Let $P(x)$ be a primitive polynomial, then we can represent the field elements as modulo $P(x)$. The example, where $P(x) = x^4 + x + 1$ is represented in Table 7 [13].

Power of ξ	**Field element**	**Vector**	**Decimal**
0	0	0000	0
ξ^0	1	0001	1
ξ^1	ξ	0010	2
ξ^2	ξ^2	0100	4
ξ^3	ξ^3	1000	8
ξ^4	$\xi + 1$	0011	3
ξ^5	$\xi^2 + \xi$	0110	6
ξ^6	$\xi^3 + \xi^2$	1100	12
ξ^7	$\xi^3 + \xi + 1$	1011	11
ξ^8	$\xi^2 + 1$	0101	5
ξ^9	$\xi^3 + \xi$	1010	10
ξ^{10}	$\xi^2 + \xi + 1$	0111	7
ξ^{11}	$\xi^3 + \xi^2 + \xi$	1110	14
ξ^{12}	$\xi^3 + \xi^2 + \xi + 1$	1111	15
ξ^{13}	$\xi^3 + \xi^2 + 1$	1101	13
ξ^{14}	$\xi^3 + 1$	1001	9

Table 7. Representation of GF(2^4) modulo $\xi^4 + \xi + 1$

After the data has been transmitted through the channel, the binary message is converted back to the non-binary form. For example if the first section of the information, that we wanted to send was 12, bit value 1100, but we received an erroneous message 1110, then in the receiver we mark it as 14 or element $\xi^3 + \xi^2 + \xi$ [8]. Because of that, the parity-check matrix of the NB LDPC code also has field elements instead of ones in it, for example

$$H = \begin{bmatrix} 0 & 0 & \xi^0 & 0 & \xi^1 & 0 & 0 & \xi^3 \\ 0 & \xi^{11} & 0 & 0 & \xi^8 & 0 & \xi^4 & 0 \\ \xi^{10} & 0 & 0 & 0 & 0 & \xi^5 & 0 & 0 \\ 0 & 0 & 0 & \xi^2 & 0 & 0 & \xi^1 & \xi^3 \end{bmatrix}.$$

If the corresponding binary parity-check matrix is needed, then every field element is replaced by the transpose of corresponding degree companion matrix [5].

8.2 Decoding

In order to decode NB LDPC codes, we use generalized belief propagation decoding. Similarly to binary LDPC decoding we have operations by rows (check nodes) and operations by columns (variable nodes), but with NB LDPC codes, variable nodes are not sending check nodes a single probability anymore, but a set of probabilities instead.

Each probability is the probability of that v_node to have one of 2^p field values. The row-code decoding is done by using BCJR algorithm, which was explained in MAP decoding using trellis paragraph and whose complexity for $GF(2^p)$ is $O(2^p \cdot 2^p)$ [8]. We use row-code trellises for decoding. Lets use the first row from a sample parity-check matrix

$$H = \begin{bmatrix} 0 & 0 & 2 & 0 & 3 & 0 & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

for NB LDPC code over $GF(2^2)$ as an example. Every row of the parity-check matrix has its own row-code trellis graph. The trellis graph has edges from every state to every next state and the edges represent the field elements. This means that the gamma values of the trellis graph are probabilities of current element being certain field element. Let k be an amount of nonzero elements in the parity-check row and q the size of the field, then there are k layers of q states in the trellis graph. To decide, which element the edge represents we use $s_i + e_n \cdot h_m$ over $GF(q)$, where s_i is the number of the state, e_n is a field element in an increasing order and h_m is the current element from the parity-check matrix [16]. The example of a row-code trellis graph for the first row of H is shown on the Figure 9.

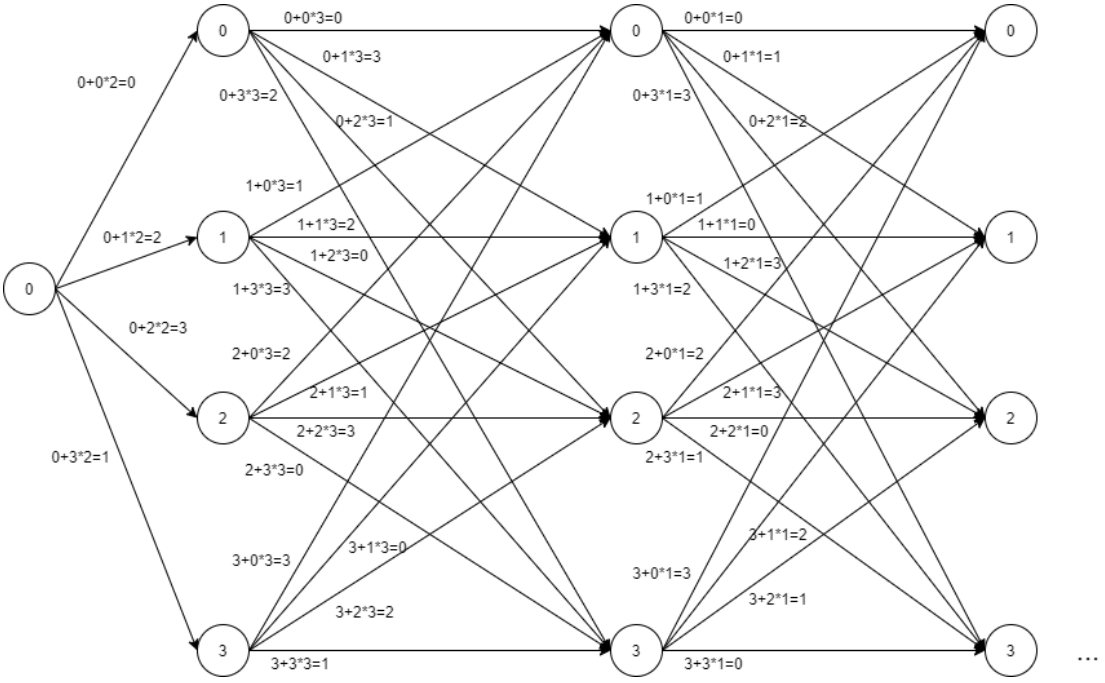

Figure 9. Row-code trellis for (0,0,2,0,3,0,0,1)

We can calculate the alpha-stage with

$$\boldsymbol{\alpha_{t+1}} = \texttt{IHT}(\texttt{HT}(\boldsymbol{\alpha_t}). * \texttt{HT}(\tilde{\boldsymbol{\gamma}}_{\boldsymbol{t}}\boldsymbol{\Pi_t}), \tag{39}$$

where HT is Hadamard transform, IHT is inverse Hadamard transform, .* is a component-wise product, $\boldsymbol{\Pi_t}$ is the permutation matrix depending on the t-th code symbol and

$$\tilde{\boldsymbol{\gamma}}_{\boldsymbol{t}} = (\gamma_t(0,0), \gamma_t(0,1), ..., \gamma_t(0,q-1)) \tag{40}$$

[6]. Analogically to alpha-stage, we can calculate beta-stage with

$$\boldsymbol{\beta_{t-1}} = \texttt{IHT}(\texttt{HT}(\boldsymbol{\beta_t}). * \texttt{HT}(\tilde{\boldsymbol{\gamma}}_{\boldsymbol{t}}\boldsymbol{\Pi_t})). \tag{41}$$

8.3 Optimization of Hadamard transform

To optimize the decoding to the coplexity of $O(2^p \cdot p)$, we use fast Hadamard transform (FHT) instead of the regular Hadamard transform (HT). The difference between HT and the FHT is that FHT takes advantage of the structure of the Hadamard matrix. Instead of multiplication with Hadamard matrix we can do additions recursively on certain elements of the input vector [11]. For example if we have a Hadamard matrix

$$H_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix},$$

a vector

$$\mathbf{v} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix},$$

and let $FHT(\mathbf{v})$ be a fast Hadamard transform over vector $\mathbf{v}$, then

$$FHT(\mathbf{v}) = \mathbf{v}H_3 = \begin{bmatrix} 5 & 1 & 1 & 1 & -1 & -1 & 3 & -1 \end{bmatrix}.$$

Figure 10 illustrates how summation using fast Hadamard transform works.

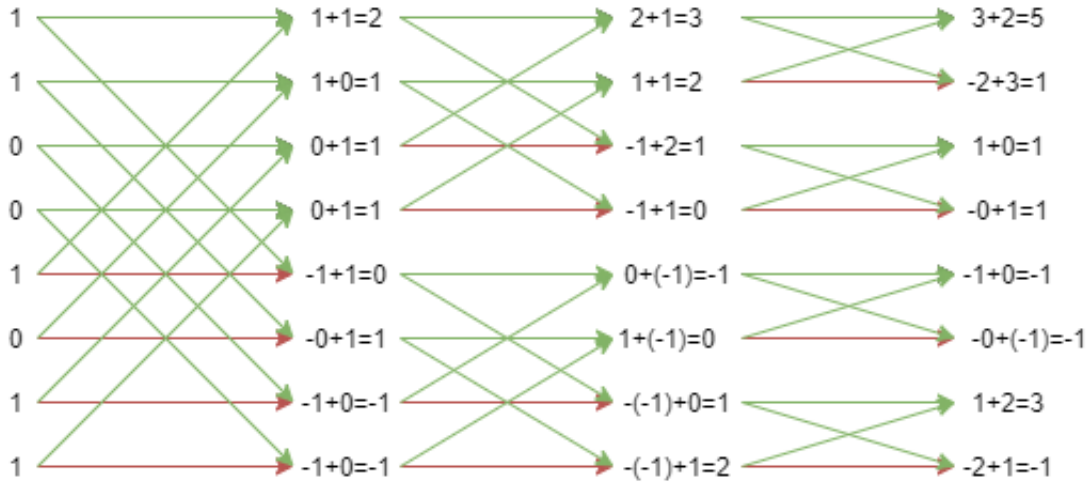

Figure 10. Fast Hadamard transform over $\boldsymbol{v}$

8.4 Matlab

The example on Figure 10 also works when moving backwards from smaller divisions to larger ones. This solution is also what we will be using in our implementation of the fast Hadamard transform. The implementation of backward fast Hadamard transform in Matlab can be seen in the code snippet below.

```
function vec_out=FHT(vec_in)
    h = 1;
    vec_out = vec_in;
    while h < length(vec_in)
        for i=1 : h*2 : length(vec_in)
            for j=i : i+h-1
                x = vec_out(j);
                y = vec_out(j+h);
                vec_out(j) = x + y;
                vec_out(j+h) = x - y;
            end
        end
        h = h*2;
    end
end
```

To compare the fast Hadamard transform with the regular Hadamard transform, we use a basic function to multiply a vector with a matrix, implemented below.

```
function out_vec=matrix_by_column_vector_mult(matrix, vector)
    [rows, columns] = size(matrix);
    out_vec = zeros(rows, 1);
    for r = 1:rows
        for c = 1:columns
            out_vec(r, 1) = out_vec(r, 1) + vector(c, 1) * matrix(r,
                c);
        end
    end
end
```

Lets compare the time differences of only the function runs for alpha-, beta- and gamma values by finding the average function speed of 100 runs. As it can be seen on Figure 11, then with the increasing of the field size, the time it takes to run HT increases exponentially with 2^p. Figure 12 shows us, that already with $GF(2^4)$ FHT is roughly two times faster than HT. Note that these times do not represent the actual times it would take for such operations to run in the real system, because Matlab is designed for testing out algorithms and is not a fast programming language. We still get the relative times for the functions.

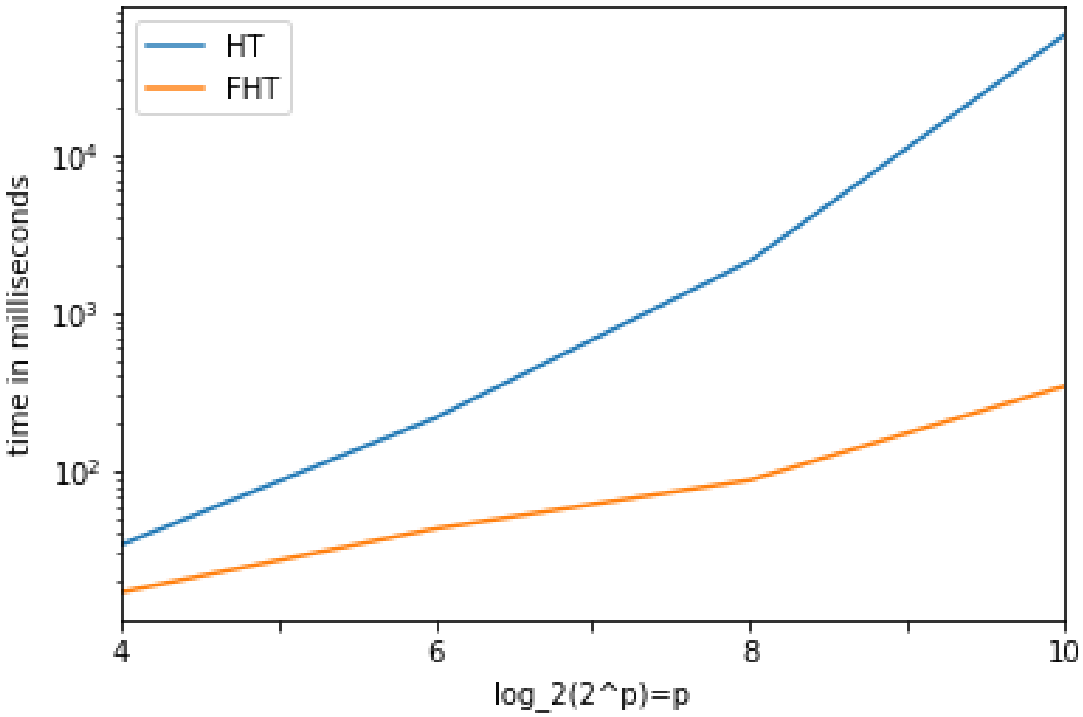

Figure 11. Function comparisons for HT and FHT for $GF(2^p)$

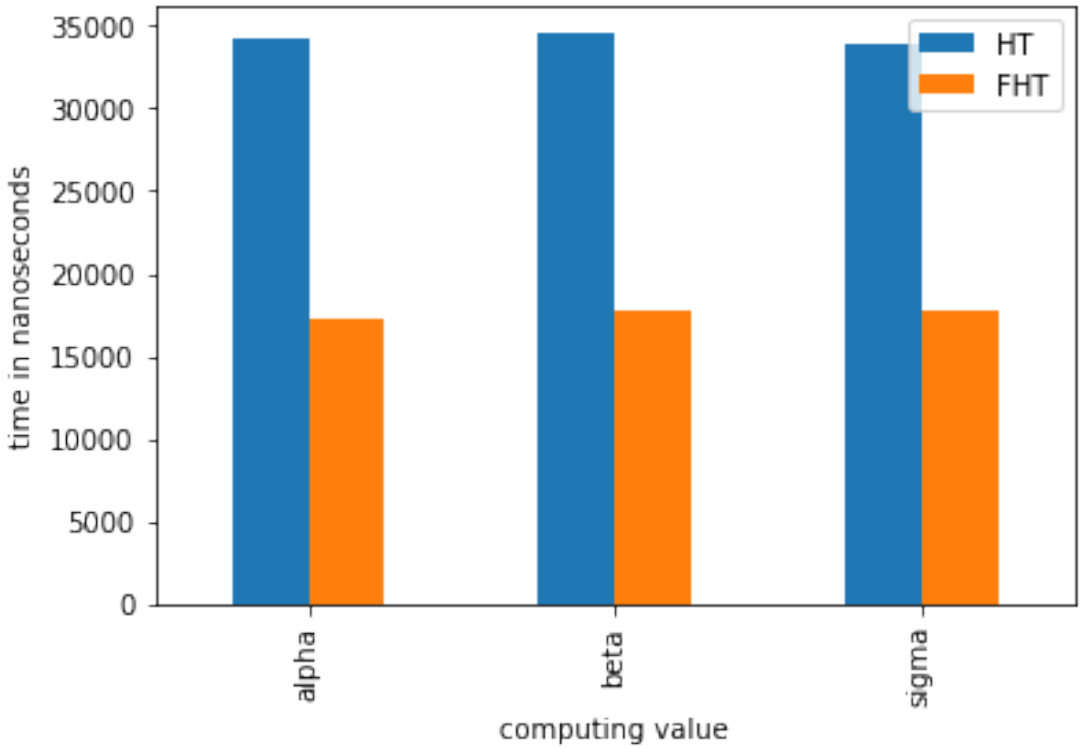

Figure 12. Function comparisons for HT and FHT for $GF(2^4)$

9 Conclusion

In this thesis, we study the domain of coding for error correction. Specifically, we focus on NB LDPC codes.

Every code structure also has its own pros and cons. For example, some decoding algorithms for binary LDPC codes are rather easy to implement and they can perform near Shannon limit, but for this, they have to be very long. On the other hand even medium and short NB LDPC codes can perform near Shannon limit, but their decoding is more complicated algorithmically.

Finally we also implemented a fast Hadamard transform and a Hadamard transform functions in Matlab and compared their performances for decoding NB LDPC codes. It was clearly seen that fast Hadamard transform is more efficient than the regular Hadamard transform, especially over larger fields.

This thesis only scratched the surface of the vast world of the channel coding, as there are more code structures and multiple decoding methods for every code structure, but NB LDPC codes are very relevant at the moment due to their efficiency in fiber-optical

and 5G technologies.

References

[1] A. Ashikhmin and S. Litsyn. “Simple MAP decoding of first-order Reed-Muller and Hamming codes”. In: *IEEE Transactions on Information Theory* 50.8 (2004), pp. 1812–1818. DOI: 10.1109/TIT.2004.831835.

[2] L. Bahl et al. “Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)” In: *IEEE Transactions on Information Theory* 20.2 (1974), pp. 284–287. DOI: 10.1109/TIT.1974.1055186.

[3] M. J. Leiner Bernhard. “LDPC Codes – a brief Tutorial”. In: (Apr. 2005).

[4] Geoffrey J Byers and Fambirai Takawira. “Fourier transform decoding of non-binary LDPC codes”. In: *Proceedings Southern African Telecommunication Networks and Applications Conference (SATNAC). Spier Wine Estate, Western Cape, South Africa*. 2004.

[5] Ben-Yue Chang, Dariush Divsalar, and Lara Dolecek. “Non-binary protograph-based LDPC codes for short block-lengths”. In: *2012 IEEE Information Theory Workshop*. 2012, pp. 282–286. DOI: 10.1109/ITW.2012.6404676.

[6] David Declercq and Marc Fossorier. “Decoding Algorithms for Nonbinary LDPC Codes Over GF(q)(q)”. In: *IEEE Transactions on Communications* 55.4 (2007), pp. 633–643. DOI: 10.1109/TCOMM.2007.894088.

[7] Ivan B. Djordjevic. “On Advanced FEC and Coded Modulation for Ultra-High-Speed Optical Transmission”. In: *IEEE Communications Surveys Tutorials* 18.3 (2016), pp. 1920–1951. DOI: 10.1109/COMST.2016.2536726.

[8] V.S. Ganepola et al. “Performance study of non-binary LDPC Codes over GF(q)”. In: *2008 6th International Symposium on Communication Systems, Networks and Digital Signal Processing*. 2008, pp. 585–589. DOI: 10.1109/CSNDSP.2008.4610743.

[9] Alain Glavieux. *Channel coding in communication networks: from theory to turbocodes*. John Wiley & Sons, 2013, "chapters 1.1, 1.2.2, 1.7, 2.1.1". URL: https://books.google.ee/books?hl=en&lr=&id=tTuJFyLTFowC&oi=fnd&pg=PP8&dq=basis+of+channel+coding&ots=smIuPHbS4y&sig=5X39TkvzcNUJF8wAvmpLWb5qAcM&redir_esc=y#v=onepage&q&f=false.

[10] Hardesty Larry. “Explained: The Shannon limit”. In: (2010). URL: https://news.mit.edu/2010/explained-shannon-0115.

[11] M. Lee and M. Kaveh. “Fast Hadamard transform based on a simple matrix factorization”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 34.6 (1986), pp. 1666–1667. DOI: 10.1109/TASSP.1986.1164972.

[12] P. Robertson, E. Villebrun, and P. Hoeher. “A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain”. In: *Proceedings IEEE International Conference on Communications ICC ’95*. Vol. 2. 1995, 1009–1013 vol.2. DOI: 10.1109/ICC.1995.524253.

[13] R. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006, "pp.2–7, 26–29, 34–35, 56–61, 458". ISBN: 9780521845045.

[14] Md. Golam Sadeque. “Bit Error Rate (BER) Comparison of AWGN Channels for Different Type’s Digital Modulation Using MATLAB Simulink”. In: *American Scientific Research Journal for Engineering, Technology, and Sciences* (June 2015).

[15] Andrew Thangaraj. *Additive White Gaussian Noise(AWGN) Channel and BPSK*. Lecture by Andrew Thangaraj. Nov. 27, 2018. URL: http://https://nptel.ac.in/courses/108/106/108106137/ (visited on 04/09/2021).

[16] Yeong-Luh Ueng et al. “A High-Throughput Trellis-Based Layered Decoding Architecture for Non-Binary LDPC Codes Using Max-Log-QSPA”. In: *IEEE Transactions on Signal Processing* 61.11 (2013), pp. 2940–2951. DOI: 10.1109/TSP.2013.2256905.

[17] E. Ryan William. “An Introduction to LDPC Codes”. In: (Aug. 2003).

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Marten Moro**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Optimized signaling for ultra-high-speed fiber-optical communications,
   *(*title of thesis)

   supervised by Vitaly Skachek and Irina Bocharova.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Marten Moro
08/05/2021