UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Uku Kangur

Fast solving of systems of linear equations in decoding of LDPC codes

Bachelor's Thesis (9 ECTS)

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

Tartu 2020

Fast solving of systems of linear equations in decoding of LDPC codes

Abstract:

In this thesis, a method is proposed, which is suitable for fast solving of sparse systems of linear equations over finite extension fields for the purpose of improving the speed of decoding related tasks for LDPC codes. The aim of the study is to optimize the Gaussian elimination algorithm for sparse systems of equations, which would solve the problem more efficiently than the standard approach. The proposed method is implemented in the Java language. Its performance is analyzed and tested experimentally.

Keywords:

System of linear equations, finite fields, sparse matrices, finite extension fields, LDPC decoding

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

Lineaarvõrrandsüsteemide kiire lahendamine LDPC koodide dekodeerimisel

Lühikokkuvõte:

Käesolev töö uurib hõredate lineaarvõrrandisüsteemide lahendamiseks mõeldud kiireid algoritme üle laiendatud korpuste LDPC koodide dekodeerimisprotsesside kiirendamise eesmärgil. Töö siht on optimiseerida Gaussi elimineermise algoritm, mis lahendaks antud probleemi kiiremini kui eelnevad alternatiivid. Pakutud meetod implementeeritakse Java keeles ning algoritmi jõudlust ning optimaalsust testitakse ning analüüsitakse eksperimentaalsete tulemuste abil.

Võtmesõnad:

Lineaarvõrrandsüsteem, korpus, laiendatud korpus, hõredad maatriksid, LDPC dekodeerimine

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

Contents

List of symbols

q	Prime number
GF(q)	Galois field of size q
GF(q^m)	Extension of the field GF(q) of degree m
C	Error-correcting code
$[n, k, d]$	Parameters of a linear code: length n, dimension k, minimum distance d
$\overline{c}$	Codeword, vector
$\overline{0}$	Vector of zeros
β	Primitive element in the field
A	Matrix A
A^{-1}	Inverse of matrix A
A^{+}	Pseudoinverse of matrix A
A^{T}	Transposed matrix A
L	Lower-triangular matrix
U	Upper-triangular matrix
H	Parity-check matrix
S	Schur compliment
ML decoding	Maximum-Likelihood decoding
BP decoding	Belief-Propagation decoding
BEC	Binary erasure channel
LDPC	Low-density parity-check
GE	Gaussian elimination algorithm
SPD	Symmetric positive definite matrix
$P(x)$	Polynomial P in the indeterminate x

r	Code rate
Pr $(\cdot\|\cdot)$	Conditional probability
Σ	Symbol alphabet
w_c	Column weight
w_r	Row weight
δ	Relative minimum distance

1 Introduction

Solving systems of linear equations is a common problem in computer science. Even though the first solutions to the problem were discovered already in the classical age (for example Gaussian elimination method was discovered in China about 2000 years ago), it made its way to Europe through the algebra books of Isaac Newton [1, 2].

In 1969, Volker Strassen found a divide-and-conquer multiplication algorithm that is faster than the Gaussian elimination (the time complexity of it being $O(n^{2.81})$) in solving systems of linear equations through the fast finding of an inverse. This discovery vastly broadened the interest in the problem and was a great scientific breakthrough in the field. Since then, mathematicians have been working to find and optimize new algorithms for this problem, since in large matrices a small change in algorithm speed can greatly improve efficiency [3].

The field of coding theory frequently uses methods based on solving systems of linear equations, especially in decoding processes. This thesis covers the methods for solving systems of linear equations used for decoding of LDPC codes and studies ways to improve their efficiency. Sparsity is a key metric here, since most conventional solving methods do not use sparsity. Our aim is to optimize Gaussian elimination to work well on LDPC codes.

In the first chapter, we cover all the required mathematical notation. The chapter gives insight into the area of finite fields, extension fields, error-correcting codes, generator matrix, parity-check matrix and minimum distance of codes. In the second chapter, we give an overview of the decoding of LDPC codes, where the solving of sparse systems of linear equations is mostly needed. The chapter also points out all of the uses of the sparse equation solving algorithm. In the third chapter, we consider improvements to solving systems of linear equations. Non-binary regular matrices are discussed first. After that, we look at the differences between algorithms for non-binary regular matrices and binary extension field matrices. In the appendix, we present a Java library for solving systems of linear equations over finite extension fields and the related decoding simulation results.

2 Terms and notation

The aim of this chapter is to make the reader familiar with the mathematical terminology and the notation used. These will serve as the foundation for the further discussion.

2.1 Finite fields

In the thesis, we use finite fields to represent systems of 2, 4, 8, 16, 32, 64, 128 and 256 elements together with operations of addition and multiplication. Since classical computers work on bits, the elements of these fields can be viewed as binary vectors. These fields can also be viewed as extentions of a binary field. Since a finite field is a special type of ring, we will start the following definition [4]:

Definition 1. A **commutative ring with unit** is a set R with two operations "$+$" and "$\cdot$" (addition and multiplication respectively) with the following properties:

(1) Associativity: $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
(2) Commutativity: $a + b = b + a$ and $a \cdot b = b \cdot a$
(3) Additive identity: there exists $0 \in R$ so that $a + 0 = a$
(4) Multiplicative identity: there exists $1 \neq 0 \in R$ so that $1 \cdot a = a$
(5) Additive inverses: For every $a \in R$, there is a additive inverse, denoted $-a$ satisfying $a + (-a) = 0$
(6) Distributivity of multiplication over addition: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$.

Definition 2. A **field** is a ring K such that every nonzero element has a multiplicative inverse. That is, for each $a \in K$ with $a \neq 0$, there is some $a^{-1} \in K$ so that $a \cdot a^{-1} = 1$.

Definition 3. A field with a finite number of elements is called a **finite field** or **Galois field** (GF).

Definition 4. A **prime field** is a field GF(q) of a prime order, which is constructed by taking integers modulo q, where q is a prime [5].

Let us take GF(7) as a prime finite field example. We can see that it is finite (it has 7 elements), has all the properties of a commutative ring and also for every $a \in$ GF(7) there exists a multiplicative inverse. This can also be seen from the addition and multiplication tables of GF(7):

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Table 1. Multiplication table of GF(7)

	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Table 2. Addition table of GF(7)

2.2 Extension fields

In this thesis we consider only the extension of the binary Galois (finite) field. These extention fields are obtained as a finite field modulo a irreducible polynomial. We work mostly over fields GF(2^2) to GF(2^8). The definiton of an irreducible polynomial is the following [5]:

Definition 5. Let F be a finite field GF(q). A polynomial $P(x) \in F[x]$, where $F[x]$ denotes the set of polynomials with the coefficients from the field F, is called **irreducible** over F if:

(i) $\deg P(x) > 0$
(ii) for any $a(x), b(x) \in F[x]$ such that $P(x) = a(x) \cdot b(x)$, either $\deg a(x) = 0$ or $\deg b(x) = 0$.

A polynomial that is not irreducible is called **reducible**.

The definiton of an extention field is the following [6]:

Definition 6. The field formed by taking polynomials over a field GF(q) modulo an irreducible polynomial $P(x)$ of degree m is called an **extension** of degree m over the field GF(q). If polynomial $P(x)$ is a primitive polynomial then all q^{m-1} nonzero elements of the extension field can be obtained as powers of its root.

As an example we present how the extention field elements are found. The following is the table of all the irreducible polynomials of degree $2, 3$ and 4 over GF(2^m):

m	Irreducible polynomials of degree m over GF(2)
2	$x^2 + x + 1$
3	$x^3 + x + 1, x^3 + x^2 + 1$
4	$x^4 + x + 1, x^4 + x^3 + 1, x^4 + x^3 + x^2 + x + 1$

Table 3. List of irreducible polynomials over GF(2)

In our software we are interested in only specific primitive polynomials for each extention degree. The following is a table of the primitive polynomials of degree $2, 3$ and 4 that we use.

n	Primitive polynomials of degree n over $\mathbb{F}_2$
2	$x^2 + x + 1$
3	$x^3 + x + 1$
4	$x^4 + x^3 + 1$

Table 4. List of primitive polynomials over GF(2) used in the software of this thesis

Example: Let us take GF(2^2). We have to find the field modulo $P(x) = x^2 + x + 1$. Let β be the root of the polynomial $P(x)$. We construct a table based on the powers of β.

Power of β	Element	Vector	Decimal
-	0	(00)	0
β^0	1	(01)	1
β^1	β	(10)	2
β^2	$\beta + 1$	(11)	3

Table 5. Representations of the extension of the binary Galois field GF(2^2) modulo $x^2 + x + 1$

2.3 Error-correcting codes

In information science and telecommunications error-correcting codes (ECC) are used for detecting and correcting errors that happen in data transmission over noisy channels. The type of error-correcting codes covered in this thesis are linear block codes, which are typically used in practice [5].

Definition 7. An (n, M, d) **block code** C is a set of $M > 0$ vectors $\overline{c}$ (called codewords) of length n over GF(q) such that $\overline{c} \in C$, where [5]:

(1) Length: n

(2) Size or cardinality: M

(3) Dimension: $k = \log_q M$

(4) Rate: $r = \frac{k}{n}$

(5) Minimum distance: d

Definition 8. Let F = GF(q) be a finite field and F^n a set of all vectors of length n over field F. The **Hamming distance** between two words $\overline{x}, \overline{y} \in F^n$ is the number of coordinates on which $\overline{x}$ and $\overline{y}$ differ. We denote the Hamming distance by $d(\overline{x}, \overline{y})$ [5].

Definition 9. The **minimum distance** of C is the minimum Hamming distance between any two distinct codewords of C; that is, the minimum distance d is given by $d = \min_{\overline{c}_1, \overline{c}_2 \in C, \overline{c}_1 \neq \overline{c}_2} \{d(\overline{c}_1, \overline{c}_2)\}$ [5].

Definition 10. A code C over a field F = GF(q) is called **linear** if C is a linear subspace of F^n over F, where F^n is a set of all vectors of length n over field F. This means that for every two codewords $\overline{c_1}, \overline{c_2} \in C$ and two scalars $a_1, a_2 \in F$ we have $a_1\overline{c}_1 + a_2\overline{c}_2 \in C$. An $[n, k, d]$ linear code is a linear code of length n, dimension k and minimum distance d. [5].

2.4 Generator matrix

Definition 11. A **generator matrix** G of a linear [n,k,d]-code is a k x n matrix whose rows form a basis of the code [5].

2.5 Parity-check matrix

Definition 12. Let C be a linear [n,k,d] code over a finite field $F = \text{GF}(q)$ and F^n a set of all vectors of length n over field F. A **parity-check matrix** H of C is an $r \times n$ matrix H over F, where $r = n - k$, such that for every $\overline{c} \in F^n$ [5]:

$$\overline{c} \in C \iff H \cdot \overline{c}^T = \overline{0}^T$$

3 Background

This chapter gives an overview of the problem background and discusses low-density parity-check code decoding on the binary and non-binary erasure channels. We will introduce the most common algorithms for solving systems of linear equations and explain how we can make decoding of LDPC codes faster.

3.1 Erasure Channels

The mathematical communications model, first introduced by Claude E. Shannon in 1948 [7], has since become the basis of all coding theory. The communication scenario consists of the source, the encoder, the channel, the decoder and the sink. This can also be seen in the following figure:

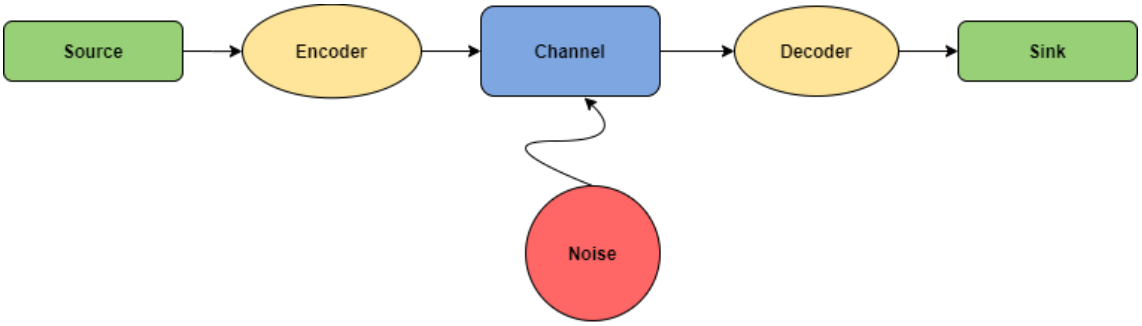

Figure 1. Communications model

The information is transferred from the source to the sink using a channel. The problem of error correction arises due to the channel noise which modifies the sent information. This is solved by error-correction in the decoder. More formally, the notion of channel is given in the next definition.

Definition 13. A channel is defined by the triple ($\sum_{in}$, $\sum_{out}$, Prob), where:

(1) $\sum_{in}$ is the input alphabet;
(2) $\sum_{out}$ is the output alphabet;
(3) Probability function Prob : $\sum_{in} \times \sum_{out} \rightarrow [0, 1]$ is defined on pairs of symbols

Prob (a, b) = Pr (b received | a transmitted),
where Pr $(\cdot|\cdot)$ denotes a conditional probability

Note that an alphabet can be either discrete or continuous. In this thesis we consider only the erasure channel. Erasure channels can be both binary or non-binary. The binary erasure channel erases each bit with probability $p \in [0, 1]$ [5]. The input and outputs are defined as $\sum_{in} = \sum_{out} = \{0, 1\}$. The probability function is defined as follows:

$$\text{Pr}\ (\ b = 1 \mid a = 1\ ) = \text{Pr}\ (\ b = 0 \mid a = 0\ ) = 1 - p$$
$$\text{Pr}\ (\ b = 1 \mid a = 0\ ) = \text{Pr}\ (\ b = 0 \mid a = 1\ ) = p$$

This is also shown in Figure 2 (where ε stands for erased element):

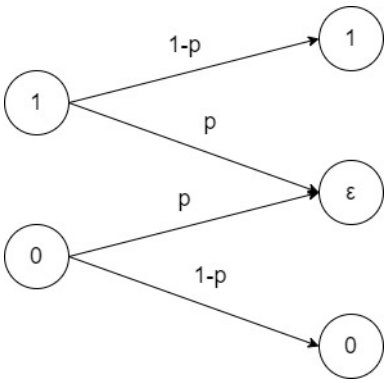

Figure 2. Binary Erasure Channel

Nonbinary LDPC codes over the extensions GF(2^m) of the binary fields demonstrate better performance than their binary counterparts [12]. They are decoded over nonbinary erasure channels where groups of m bits corresponding to nonbinary symbols are erased.

3.2 LDPC codes

The main code family, which is considered in this thesis, are the low-density parity-check (LDPC) codes. LDPC codes were first introduced by Robert G. Gallager in his doctorial thesis [8]. The main advantage of LDPC codes is existence of low-complexity (linear with the code length) iterative decoding procedures. LDPC codes functionally are defined by a sparse parity-check matrix. Let w_r be the average number of non-zero elements in a row and w_c the average number of non-zero elements in a column. For a code defined by its $(n-k) \times n$ parity-check matrix (where $(n-k)$ is the number of rows and n the number of columns) to be considered low density, the following two conditions have to hold: $w_c << (n-k)$ and $w_r << n$ [9]. Even though regular LDPC codes are not the best error-correcting codes in the sense of the correcting capability, they are asymptotically good. This means that if $w_c \geq 3$, then minimum distance for this class of codes grows linearly with their length [10]. More formally if $n \to \infty$, we have $k/n \to r > 0$ and $d/n \to \delta > 0$. In a code where $w_c = 2$ and $w_r = 4$, a parity-check matrix looks for example as follows:

$$H_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

To have a large enough minimum distance one should use rather long LDPC codes. In practical cases codes can reach the size of over 10000 symbols.

LDPC codes can be both binary or non-binary. In non-binary cases the non-zero elements are usually defined over binary extention fields (it is easy to imagine these symbols as packets of m bits if GF(2^m) is used). In this thesis, for the sake of implementation convenience, we associate the symbols in the extension field GF(2^m) with the corresponding polynomials with binary coefficients, which, in turn, are associated with decimals numbers. For example, the polynomial $x^2 + 1$ is associated with the binary vector 101, which is, in turn, is associated with the decimal number 5. A non-binary version of the matrix H over GF(2^4) could be written as follows:

$$H_2 = \begin{bmatrix} 8 & 10 & 0 & 0 & 0 & 14 & 0 & 0 & 0 & 3 \\ 12 & 0 & 0 & 6 & 0 & 0 & 1 & 8 & 0 & 0 \\ 0 & 2 & 0 & 0 & 3 & 0 & 11 & 0 & 4 & 0 \\ 0 & 0 & 7 & 0 & 9 & 0 & 6 & 10 & 0 & 8 \\ 0 & 0 & 2 & 1 & 0 & 7 & 0 & 0 & 14 & 0 \end{bmatrix}$$

The rows and columns of the parity-check matrix of an LDPC code can have different weights. The LDPC code defined by such a parity-check matrix is called irregular. We call a LDPC code regular if w_c is constant for every column and $w_r = w_c \cdot (n/(n-k))$ is also constant for every row [9].

Another way to view LDPC codes is by using a Tanner graph, which was introduced in 1981 by R. Michael Tanner [11]. Tanner graph of a parity-check matrix is a bipartite graph whose biadjecency matrix is H. It consists of column nodes v (also called variable nodes) and row nodes c (also called check nodes). The connections of these nodes mark the positions of non-zero elements in the matrix. The following is a Tanner graph representation of the previously shown LDPC code (H_1):

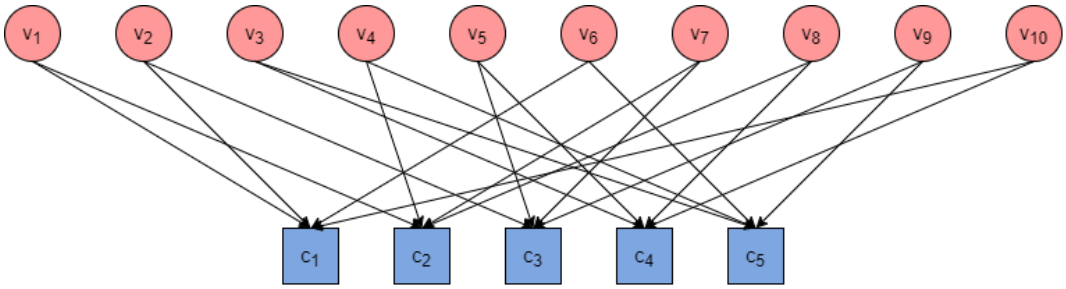

Figure 3. Tanner Graph of matrix H_1

There are two common methods for decoding of LDPC codes:

(1) Belief-propagation (BP) decoding. This is commonly used low-complexity iterative decoding technique. Its main shortcoming is that the BP decoding performance is not optimal.

(2) Maximum-likelihood (ML) decoding. This is an optimal decoding, which minimizes the decoding error probability. However, its computational complexity in soft-decision channels is exponential and for this reason it is not applied to long LDPC codes. Fortunately, in erasure channel ML decoding of error-correcting codes can be reduced to solving system of linear equations and has at most cubic (with the code length) complexity. In this thesis we optimize ML decoding of LDPC codes in erasure channel by taking into account sparsity of their parity-check matrices. Notice, that BP decoding over erasure channel is also simplified to so-called peeling algorithm discussed in the next section.

3.3 Decoding of LDPC codes over erasure channel

LDPC codes with iterative decoding are used in most of the existing communication standards. Nonbinary LDPC codes allow for improved decoding performance compared to their binary counterparts. However, performance of iterative decoding (so-called belief propagation decoding) even for nonbinary LDPC codes is still inferior to the theoretically optimal performance [12].

In this thesis we focus on the decoding of LDPC codes over the erasure channel. In fact, decoding over noisy communication channels can be reduced to decoding over erasure channel by erasing unreliable code symbols [13]. On the other hand, in larger network applications, the lost data packets are often interpreted as erasures [14]. This makes the erasure channel an important communication model.

Due to the above reasons the decoding can be reduced to a problem of solving a system of linear equations over a finite field. Sparsity of the obtained system allows for iterative belief-propagation based peeling decoding algorithm for solving the system of linear equations [15]. The idea behind this algorithm is based on iterative solving of equations with only one unknown until such equations exist. This can also be seen in the following general algorithm for erasure channel decoding example [16]:

Algorithm 1: Iterative decoding on erasure channel

Input: Tanner graph of H and received word with erasures
$\overline{y} = (y_1, \ldots, y_n) \in (\text{GF}(q) \cup \{\varepsilon\})^n$

Output: Decoded codeword $\overline{c} = (c_1, \ldots, c_n) \in (\text{GF}(q))^n$

Step 1. The values of symbols y_i from variable nodes v_i are sent to the check nodes.

Step 2. If there is no check node which has received exactly one erased position ε, then the decoding algorithm stops due to the decoding being impossible.

Step 3. Otherwise let c_j be the check-node that received exactly one ε (in case there are several of such nodes, pick any one of them) and let v_i be the variable node, that has sent ε to c_j. Then from parity-check equation we can find that the value in v_i. We can see that this value is equal to the sum of the other values received by c_j.

Step 4. The deduced value is sent back to the variable node v_i.

Step 5. Repeat steps 2. to 4. until all the erasures are restored or a decoding error happens. If all the erasures are restored, then output decoded message.

The success probability of such an iterative decoding is not optimal. The optimal maximum-likelihood solution can be obtained with time complexity of order of $O(n^3)$ by applying Gaussian elimination. There is a need in more efficient techniques which would take into account sparsity of the system.

We also present the comparison of both BP and ML decoding simulations in the Appendix 1 and 2, where we compare these in relation to the error rate. There are two error rates that we measure:

(1) FER – frame error rate. It measures the probability that the frame (the codeword) was not decoded correctly.

(2) BER – bit error rate. It measures the probability that a bit or symbol was not decoded correctly. Even if the frame is not decoded correctly, many bits are usually correct. Therefore, the BER is usually smaller than FER.

3.4 Overview of methods for solving systems of linear equations

There are a lot of different methods that solve systems of linear equations. This chapter gives an overview of some of the existing methods for solving linear system of equations and the related operations.

3.4.1 Solving system of linear equations using matrices

A system of linear equations is a set of linear equations that use the same variables (unknowns). In computer systems a common way to view systems of linear equations are in their augmented matrix form. An example of a reformatted system of linear equations is as follows:

$$\begin{aligned} 3 \cdot x_1 + 2 \cdot x_2 - 1 \cdot x_3 &= 2 \\ 5 \cdot x_1 + 1 \cdot x_2 + 2 \cdot x_3 &= 0 \\ 4 \cdot x_1 + 2 \cdot x_2 - 1 \cdot x_3 &= 1 \end{aligned}$$

$$\left[\begin{array}{ccc|c} 3 & 2 & -1 & 2 \\ 5 & 1 & 2 & 0 \\ 4 & 2 & -1 & 1 \end{array}\right]$$

In order to solve this system we need to find for $\overline{x}$ in the equation $A\overline{x} = \overline{b}$, where A is the matrix containing coefficients of unknowns, $\overline{x}$ is the vector of unknowns and $\overline{b}$ is the value vector of all linear equations in the system. The previous example would look as follows:

$$\underbrace{\begin{bmatrix} 3 & 2 & -1 \\ 5 & 1 & 2 \\ 4 & 2 & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\overline{x}} = \underbrace{\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}}_{\overline{b}}$$

In the sequel, we consider ML decoding of non-binary LDPC codes defined by parity-check matrices of size $(n-k) \times n$. In order to simplify implementation of ML decoding, we first decode the received vector by Algorithm 1 (peeling decoding) and then solve the system of linear equations obtained after applying Algorithm 1.

Belief-propagation decoding might fail to fully restore all the erased symbols. In that case, the remaining erased values of the codeword symbols together with the corresponding columns of the parity-check matrix and the syndrome vector form a linear system of equations. Next, consider a system of l linearly independent equations with s unknowns. If $s > l$ or $s = l$, which is a typical case, there is a solution to the system. In the case when $s < l$, there is no solution.

The algorithms for solving systems of linear equations group into the following two method families:

1. **Direct method** algorithms compute the solution to a problem in a finite number of steps. That means that the output of these algorithms is always a precise answer.

2. **Iterative method** algorithms in contrast, start from an initial guess and iteratively converge to an exact solution. Iterative methods are much more time efficient when it comes to large matrices, but they lose in precision.

In the following table, we list some of the known algorithms for this problem:

Algorithm	Method	Matrix structure	Year	Speed (FLOPs)	Author
Gaussian elimination	direct	any	-	$2s^3/3$	-
Cramer's rule	direct	$s = l$	1750	$s! \times s$	Cramer [17]
Cholesky decomposition	direct	SPD	1910	$s^3/3$	Cholesky [18]
LU-decomposition	direct	any	1948	$2s^3/3$	Turing [20]
QR-decomposition	direct	any	1961	$4s^3/3$	Francis [21][22], Kublanovskaya [23]
Golub-Reinsch (SVD)	direct	any	1970	$6s^3$	Golub & Reinsch [24]
Least squares	iterative	$s > l$	1795	$a \times i^*$	Gauss [25], Legende [26]
Jacobi	iterative	SPD	1845	$a \times i^*$	Jacobi [27]
Gauss-Seidel	iterative	SPD	1872	$a \times i^{**}$	Seidel [28]
Richardson	iterative	SPD	1910	$a \times i^*$	Richardson [29]
Method of steepest descent	iterative	SPD	1939	$a \times i^*$	Temple [30]
SOR	iterative	SPD	1950	$a \times i^*$	Young [31], Frankel [32]
Conjugate gradient	iterative	SPD	1952	$a \times i^*$	Hestenes & Stiefel [33]

Table 6. Algorithms for solving systems of linear equations

* - The runtime depends on the number of iterations, therefore time complexity is $O(ai)$, where a is the number of iterations and i the complexity of one iteration.
SPD stands for symmetric positive definite matrices.

Note that most of these algorithms do not explicitly work on sparse matrices, thus it is interesting to investigate whether some algorithms can work more efficiently with sparse matrices. This is why it is important to mention and analyze them, when looking for most optimal methods for solving sparse systems. The algorithm selection process for sparse systems is discussed later on in Chapter 3.4.5.

When solving systems of linear equations using direct method algorithms, there are two tehniques:

1. Using elementary row-operations to achieve row-echelon format and back-substituting the answer. This is the standard method of Gaussian elimination.

2. Since $A\overline{x} = \overline{b}$, then we can see that $\overline{x} = A^{-1}\overline{b}$. This means that we can find the inverse of A and multiply it by $\overline{b}$ to get the solution vector $\overline{x}$. Finding an inverse of a matrix depends on the speed of multiplication of two $s \times s$ matrices thanks to the blockwise invertion (or analogous solutions, like the Strassen algorithm).

3.4.2 Matrix inversion

Definition 14. A $s \times s$ matrix A is called invertible (nonsingular) if there exists a $s \times s$ matrix B, such that $AB = BA = I_s$.
I_s is a identity matrix of size s in this case.

Definition 15. A **pseudoinverse** (also known as Moore-Penrose inverse) of a $s \times l$ matrix A over the field GF(q) is defined as a $s \times l$ matrix B over the field GF(q) such that all of the Moore-Penrose [34][35] conditions are satisfied:
1. $ABA = A$
2. $BAB = B$
3. $(AB)^* = AB$
4. $(BA)^* = BA$
Note that * marks the Hermitian transpose of a matrix.

When A is non-singular, any generalized inverse $B = A^{-1}$ is unique, but in all other cases, there are an infinite number of matrices that satisfy the first condition. However the pseudoinverse is unique [36]. Therefore we are only interested in non-singular inverses and pseudoinverses. We denote the inverse of the matrix A as A^{-1} and pseudoinverse as A^{+}.

There are different methods for finding the inverse of a matrix. The following table gives an overview of some known methods:

Algorithm	Method	Output	Year	Complexity	Author
Gaussian elimination	direct	inverse	-	$O(s^3)$	-
Newton	iterative	inverse	1669	$O(a \times i)$	Newton [37]
Cramers rule	direct	inverse	1750	$O(s! \times s)$	Cramer [17]
Least squares	iterative	pseudoinverse	1795	$a \times i$	Gauss [25], Legende [26]
Cholesky decomposition	direct	inverse, pseudoinverse	1910	$O(s^3)$	Cholesky [18]
Blockwise inversion	direct	inverse	1937	$O(f(s)^*)$	Banachiewicz [19]
LU-decomposition	direct	inverse, pseudoinverse	1948	$O(s^3)$	Turing [20]
QR-decomposition	direct	inverse, pseudoinverse	1961	$O(s^3)$	Francis [21][22], Kublanovskaya [23]
Strassen	direct	inverse	1969	$O(s^{2.81})$	Strassen [38]
Golub-Reinsch (SVD)	direct	pseudoinverse	1970	$O(s^3)$	Golub & Reinsch [24]

Table 7. Algorithms for finding inverse of matrix A

* - $f(s)$ in this case is the complexity of multiplying two $s \times s$ matrices.

From this table we can see that the fastest method Blockwise inversion depends on the speed of multiplying matrices (full proof shown in [39]). As an example, let us imagine we have a $s \times s$ matrix X ($X\overline{x} = \overline{b}$), where we split the matrix into four $s/2 \times s/2$ sized blocks A, B, C, D.

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Let $S = D - CA^{-1}B$ be the Schur compliment of X. Then we can get the inverse X^{-1} as follows:

$$X^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{bmatrix}$$

The final solution $\overline{x}$ can be achieved through solving $\overline{x} = A^{-1}\overline{b}$ as introduced previously.

3.4.3 Matrix multiplication

The time complexity of fast methods for solving systems of linear equations rely heavily on the speed of multiplication of two $s \times s$ matrices. The time complexity of the algorithms for solving a system of s equations with $O(s)$ unknowns is believed to be $O(s^w)$, where $w > 0$ is some constant. On one hand, the Gaussian elimination method requires time $O(s^3)$. On the other hand, the input to the problem contains $\Omega(s^2)$ values, and therefore $\Omega(s^2)$ is the lower bound on the time complexity. Thus, the optimal w is between 2 and 3. It is possible to compose a table for historical improvement of the achieved exponent for solving systems of linear equations.

Year	Exponent	Author
<1969	3	-
1969	2.81	Strassen [38]
1978	2.79	Pan [40]
1979	2.78	Pan [41]
1979	2.78	Bini et al [42]
1981	2.55	Schönhage [43]
1981	2.53	Pan [44]
1982	2.52	Romani [45]
1982	2.50	Coppersmith and Winograd [46]
1986	2.48	Strassen [47]
1987	2.376	Coppersmith and Winograd [48]
2012	2.3729	Williams [50]
2014	2.3728639	Le Gall [3]

Table 8. Historical improvement on the achieved exponent for solving systems of linear equations

The fastest known matrix multiplication algorithm, that is based on 1987 Coppersmith-Winograd algorithm was proposed by Francois Le Gall in 2014 and has the time com-

plexity $O(s^{2.3728639})$ [3].

3.4.4 Matrix decompositions

One of the most common way for solving sparse systems of equations is based on using matrix decomposition algorithms. The fastest decomposition algorithms are at the same time complexity as Gaussian elimination ($O(s^3)$). In these systems, if the decomposition is formed, then the elimination time complexity is only $O(s^2)$ for different value vectors $\overline{b}$ (in $A\overline{x} = \overline{b}$). This is useful if the matrix A does not change during different runs. In many applications however, this is not the case.

As an example of a decomposition based solving, we will present the LU-decomposition. LU-decomposition of the matrix A is a product of the lower-triangular matrix L (which has all ones on the main diagonal) and upper-triangular matrix U. This can be seen from the following equation:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{21} & 1 & 0 & 0 \\ L_{31} & L_{32} & 1 & 0 \\ L_{41} & L_{42} & L_{43} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}$$

This form can be achieved by using Gaussian elimination to find L and based on that we can find U. The following is an example for solving a system of linear equations by using the LU-decomposition (through inversion):

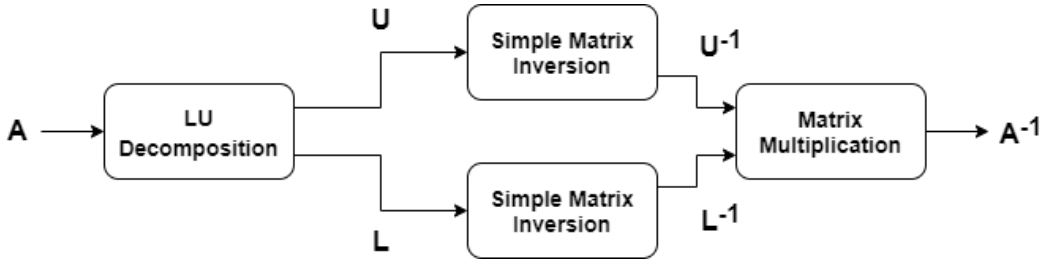

Figure 4. Getting an inverse of matrix A using LU-decomposition [51]

We can also solve the decomposition directly by using the forward and backward substitution. For example in LU-decomposition, if we have the lower (L) and upper triangular matrices (U), then we can solve $L\overline{y} = \overline{b}$ and after that solve $U\overline{x} = \overline{y}$, where we get the values of $\overline{x}$. The complexity of this is only $O(s^2)$.

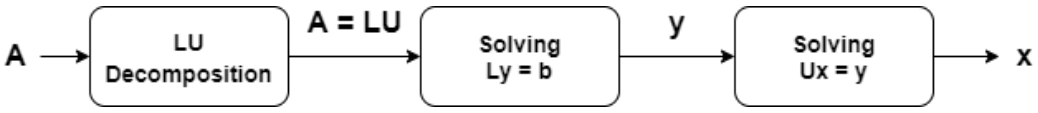

Figure 5. Solving $A\overline{x} = \overline{b}$ by substitution using LU-decomposition [20]

3.4.5 Solving systems of linear equations for LDPC codes

When trying to solve systems of linear equations, we can disregard iterative methods, since these methods are good asymptotically and do not perform better than Gaussian elimination in systems of size $s < 100$, which is the case considered wherein. We can also disregard algorithms that require $s = l$ or symmetric matrices, since the systems of equations can also have $s < l$ or $l < s$. We have chosen to optimize Gaussian elimination since it suits well the sparse systems. By contrast in many of the other algorithms, there are many substeps that make optimizing for sparseness not as efficient.

3.4.6 Gaussian elimination

Gaussian elimination is the most known and widely used algorithm for solving systems of linear equations. Consider a system of linear equations with s equations and l unknowns, namely $A\overline{x} = \overline{b}$. In the matrix form, such a system has a form [1]:

$$\underbrace{\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\overline{x}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\overline{b}}$$

In this approach, first, the matrix is turned into the row-echelon form. By using elementary row operations it is possible to obtain all zeros under the main diagonal of A.

Elementary row operations are either the swapping of rows, the multiplication of an equation with a non-zero integer or the addition of an equation with another already multiplicated equation. In this example the different constant coefficients of multiplications have been marked as c, c' and c''. After completing this step the result is the following matrix [1]:

$$\underbrace{\begin{bmatrix} c_{11}a_{11} & \dots & c_{1n}a_{1n} \\ 0 & \ddots & \vdots \\ 0 & 0 & c_{mn}a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c'_1 x_1 \\ \vdots \\ c'_n x_n \end{bmatrix}}_{\overline{x}} = \underbrace{\begin{bmatrix} c''_1 y_1 \\ c''_2 y_2 \\ \vdots \\ c''_n y_n \end{bmatrix}}_{\overline{b}}$$

From here, a backward substitution must be used to obtain values of all of the unknowns. Because we need to make the elements under the main diagonal equal zeros, and there are s^2 such elements, each elementary row operation required time $O(s)$. Therefore the total time complexity of the Gaussian elimination method is $O(s^3)$, which makes it very inefficient when it comes to solving large systems of linear equations [1].

4 Solving sparse systems of linear equations over finite extention fields

In this subsection, we consider sparse systems of linear equations over finite extension fields. The elimination process is different due to the specific arithmetical propreties of the fields. The extention fields we work with here range from GF(2^2) to GF(2^8), but these methods can be used on larger fields as well. The systems are sparse and thus Gaussian elimination in it's original form, is very inefficient, since there are a lot of zero elements and doing operations on zeros can take up computation time. Therefore we propose an algorithm that gives a more efficient approach to this

4.1 Constructing the extention field

Before any elimination over extension fields can be done, the arithmetic has to be defined. For this we construct a table of the field elements modulo polynomial $P(x)$. This polynomial $P(x)$ together with the field extention GF(2^m) and β (the root and solution to $P(x)$) are what define the table. Let a field GF(2^4) be defined by the primitive polynomial $P(x) = x^4 + x^3 + 1$. Let β be the root of the polynomial $P(x)$. Therefore the root β satisfies: $\beta^4 = \beta^3 + 1$. The following would be the table of elements over the extension field:

Power of β	Element	Vector	Decimal
-	0	(0000)	0
β^0	1	(0001)	1
β^1	β	(0010)	2
β^2	β^2	(0100)	4
β^3	β^3	(1000)	8
β^4	$\beta^3 + 1$	(1001)	9
β^5	$\beta^3 + \beta + 1$	(1011)	11
β^6	$\beta^3 + \beta^2 + \beta + 1$	(1111)	15
β^7	$\beta^2 + \beta + 1$	(0111)	7
β^8	$\beta^3 + \beta^2 + \beta$	(1110)	14
β^9	$\beta^2 + 1$	(0101)	5
β^{10}	$\beta^3 + \beta$	(1010)	10
β^{11}	$\beta^3 + \beta^2 + 1$	(1101)	13
β^{12}	$\beta + 1$	(0011)	3
β^{13}	$\beta^2 + \beta$	(0110)	6
β^{14}	$\beta^3 + \beta^2$	(1100)	12

Table 9. Elements over extension finite field of GF(2^4) modulo $\beta^4 + \beta^3 + 1$

4.2 Creating calculation tables

There are different ways to represent elements of the extension field which are polynomials. First way is to represent polynomials as binary vectors of their coefficients and the second is to consider decimal equivalents of the corresponding binary vectors. Depending on if we use polynomials or numerical representations, the arithmetic can be implemented differently. In the first case we can add together the polynomials modulo 2, but with bigger extension fields the performance does not scale well. Another way for implementing addition over binary extension fields is taking the exclusive disjunction (XOR) of either the binary or decimal element representations. This is a better way, since we only need to compare each bit once and do not need to do any additional operations. Examples of both ways over the previously shown extension field look as follows:

$$\beta^7 + \beta^{11} = (\beta^2 + \beta + 1) + (\beta^3 + \beta^2 + 1) = \beta^3 + 2 \cdot \beta^2 + \beta + 2 \cdot 1 = \beta^3 + \beta = \beta^{10}$$

Alternatively,

$$\beta^7 + \beta^{11} = 0111 \oplus 1101 = 1010 = \beta^{10}$$

Multiplication over binary extension fields can also be done in different ways. One way is to multiply the polynomials together over modulo 2 with reducing modulo primitive polynomial. Another way is to add together the powers of β over modulo $2^m - 1$ where m is the extention degree. Example:

$$\beta^7 \cdot \beta^{11} = (\beta^2 + \beta + 1) \cdot (\beta^3 + \beta^2 + 1) =$$
$$\beta^5 + \beta^4 + \beta^2 + \beta^4 + \beta^3 + \beta + \beta^3 + \beta^2 + 1 = \beta^5 + \beta + 1 = \beta^3 + \beta + 1 + \beta + 1 = \beta^3$$

Alternatively,

$$\beta^7 \cdot \beta^{11} = \beta^{7+11} = \beta^{18} = \beta^{15} \cdot \beta^3 = 1 \cdot \beta^3 = \beta^3$$

Inverse of an element is easy to find when we know how to do multiplication. If we want to find an inverse of an element a, then. we have to look for another element b, where $a \cdot b = 1$. For example:

$$(\beta^{12})^{-1} = \beta^{-12} = \beta^{-12} \cdot \beta^{15} = \beta^3$$

After selecting arithmetic methods, the calculation tables can be formed for all pairs of elements as in Tables 8 and 9 (note that the elements are held here as decimals for simplicity).

It is not optimal to build an addition table since taking XOR of two decimals is a fast process by itself. Instead of an inversion table, a flipped multiplication table can also be used (so called division table).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	4	6	8	10	12	14	9	11	13	15	1	3	5	7
3	0	3	6	5	12	15	10	9	1	2	7	4	13	14	11	8
4	0	4	8	12	9	13	1	5	11	15	3	7	2	6	10	14
5	0	5	10	15	13	8	7	2	3	6	9	12	14	11	4	1
6	0	6	12	10	1	7	13	11	2	4	14	8	3	5	15	9
7	0	7	14	9	5	2	11	12	10	13	4	3	15	8	1	6
8	0	8	9	1	11	3	2	10	15	7	6	14	4	12	13	5
9	0	9	11	2	15	6	4	13	7	14	12	5	8	1	3	10
10	0	10	13	7	3	9	14	4	6	12	11	1	5	15	8	2
11	0	11	15	4	7	12	8	3	14	5	1	10	9	2	6	13
12	0	12	1	13	2	14	3	15	4	8	5	9	6	10	7	11
13	0	13	3	14	6	11	5	8	12	1	15	2	10	7	9	4
14	0	14	5	11	10	4	15	1	13	3	8	6	7	9	2	12
15	0	15	7	8	14	1	9	6	5	10	2	13	11	4	12	3

Table 10. Multiplication table

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	1	12	8	6	15	4	14	3	13	11	10	2	9	7	5

Table 11. Inversion table

4.3 Gaussian elimination over extension field

Gaussian elimination over an extension field for the most part follows the same steps, but uses the new field arithmetic. Take the following solution over the field GF(2^4) as an example:

$$\begin{bmatrix} 0 & 7 & 14 \\ 6 & 4 & 6 \\ 5 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 6 \end{bmatrix}$$

We start from the first element of the first row and make that the pivot. First we have to make sure that the pivot we select is a non-zero element, therefore we swap it with a non-zero row.

$$\begin{bmatrix} 6 & 4 & 6 \\ 0 & 7 & 14 \\ 5 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 6 \end{bmatrix}$$

Now we eliminate down all of the remaining elements in the first column. As the first column element in the second row is already zero, then we can focus on the third row. We have to find an element α, such that $5 + \alpha \times 6 = 0$. Using the multiplication table, we can see that α is 13, since $5 + 13 \times 6 = 0$. Now we can multiply the first row by 13 and add it to the third row. We obtain the following:

$$\begin{bmatrix} 6 & 4 & 6 \\ 0 & 7 & 14 \\ 0 & 7 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 8 \end{bmatrix}$$

Now we shift the pivot to the next row and second element. We do the same processes as before and iterate the steps for all rows of the matrix until we have obtained the row-echelon form:

$$\begin{bmatrix} 6 & 4 & 6 \\ 0 & 7 & 14 \\ 0 & 0 & 15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 13 \end{bmatrix}$$

From this form we can back-substitute the valued of known variables by using the calculation tables and find the solution:

$$15 \cdot x_3 = 13 \implies x_3 = 11$$
$$7 \cdot x_2 + 14 \cdot 11 = 5 \implies x_2 = 11$$
$$6 \cdot x_1 + 4 \cdot 11 + 6 \cdot 11 = 3 \implies x_1 = 2$$

This example covers only a unique solution problems, but systems of linear equations can also have multible solutions, for example:

$$\begin{bmatrix} 6 & 4 & 6 \\ 0 & 7 & 14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

For these situations we can either return the basis of the solution space or output all the solutions. Note that in larger systems the solution space can get very large and has to be limited.

4.4 Gaussian eliminaton of sparse matrices over extention fields

As it was stated previously, using standard Gaussian elimination for solving systems of linear equations is not optimal in terms of time complexity. Thus we exploit the availability of zero elements by using different data structures. The algorithm used to solve these issues is as follows:

Algorithm 2: Optimized Gaussian elimination

Input: matrix A, syndrome vector $\overline{b}$, sum table, product table, inversion table.

Output: solution vector $\overline{x}$.

Step 1. Get all of the positions of non-zero elements (by index) and store them into row and column tables.

Step 2. Check if the pivot is a non-zero element, if not then change the row with another where element in the current column is non-zero.

Step 3. Reduce all the non-zero elements to zeros by eliminating using the pivot. Update the column and row tables.

Step 4. Continue on to the next pivot (next row and column). Repeat Steps 2, 3 and 4 until row echelon form is achieved.

Step 5. Do back-substitution by going through only the non-zero elements by each row.

Step 6. Output the solution.

Consider matrix A and a vector $\overline{b}$. The system is as follows:

$$A\overline{x} = \begin{bmatrix} 12 & 0 & 0 & 0 & 0 & 1 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 14 & 0 & 0 & 4 & 0 & 0 \\ 0 & 4 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 10 \\ 9 \\ 0 \\ 0 \end{bmatrix} = \overline{b}$$

First we create the row and columns arrays with non-zero element positions.

columns = [[0,2],[1,3],[1,3],[2,4],[4,5],[0,5]]

rows = [[0,5],[1,2],[0,3],[1,2],[3,4],[4,5]]

We start from the upper-left element of A and pick it as the pivot. Then we check if the pivot is zero. Since 12 is a non-zero element, then we do not have to switch the rows. Now, we start eliminations as it is described in the last chapter but rather than going

through all the elements, we only need to check the non-zero elements in the column. We can get the position from the column array that we constructed beforehand. In this case the next non-zero element 14 is at position **2** of the first column.

columns = [[0,**2**],[1,3],[1,3],[2,4],[4,5],[0,5]]

After finding the non-zero element row, we eliminate the pivot row from the selected row. Since we want to ignore the zeros in the rows as well, then we only operate at the positions, where the pivot row has non-zero elements. These elements are also highlighted in the following:

rows = [**[0,5]**,[1,2],[0,3],[1,2],[3,4],[4,5]]

$$A\overline{x} = \begin{bmatrix} \mathbf{12} & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 11 & 3 & 0 & 0 & 0 \\ \mathbf{14} & 0 & 0 & 4 & 0 & \mathbf{0} \\ 0 & 4 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 10 \\ 9 \\ 0 \\ 0 \end{bmatrix} = \overline{b}$$

After we eliminate the elements in the first column (and the corresponding row element in the value vector $\overline{b}$), the system looks as following:

$$A\overline{x} = \begin{bmatrix} \mathbf{12} & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 11 & 3 & 0 & 0 & 0 \\ \mathbf{0} & 0 & 0 & 4 & 0 & \mathbf{6} \\ 0 & 4 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 1 \\ 9 \\ 0 \\ 0 \end{bmatrix} = \overline{b}$$

After that, we update the row and column arrays by checking the elements only at the given indices.

rows [[0, 5], [1, 2], [0, 3], [1, 2], [3, 4], [4, 5]]
columns [[0, 2], [1, 3], [1, 3], [2, 4], [4, 5], [0, 5]]

When all of the elements are eliminated in the pivot column, we choose a new pivot which is one position to the right and to the bottom. The new pivot is marked in bold:

$$A\overline{x} = \begin{bmatrix} 12 & 0 & 0 & 0 & 0 & 1 \\ 0 & \mathbf{11} & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 6 \\ 0 & 4 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 1 \\ 9 \\ 0 \\ 0 \end{bmatrix} = \overline{b}$$

We repeat the same process with the new pivot. Note that since we only care about the elements below the pivot row, then we can ignore the rows above it when iterating through the column non-zero elements. We repeat this until we reach the row-echelon form:

$$A\overline{x} = \begin{bmatrix} 12 & 0 & 0 & 0 & 0 & 1 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 6 \\ 0 & 0 & 0 & 0 & 2 & 5 \\ 0 & 0 & 0 & 0 & 0 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 9 \\ 1 \\ 8 \\ 12 \end{bmatrix} = \overline{b}$$

To obtain the values of all entries in $\overline{x}$ we have to back-substitute the known valued of $\overline{x}$. Note that we substitute only with the non-zero elements by using existing row and column arrays. We obtain the following answer:

$$\overline{x} = [11, 9, 4, 9, 8, 9]$$

The program outputs the answer as an integer array, and the execution of the algorithm is finished.

4.5 Complexity and performance

We analyse the optimized Gaussian elimination by the different steps it takes. First the algorithm creates row and column arrays for non-zero elements, then it does the elimination into row-echelon form and from there it back-substitutes to get the values of unknowns. It is known that ultra-sparse non-binary LDPC codes perform very well in practice [52]. When the matrix A is obtained from such a non-binary LDPC code, for example from an LDPC code with a column weight 2, then it can be shown that the optimized GE algorithm performs only $O(s^2)$ operations. In that case, we obtain a significant improvement in complexity compared to the regular Gaussian elimination ($O(s^3)$). The performance of each of these steps for GF(2^4) can also be seen from Figure 6.

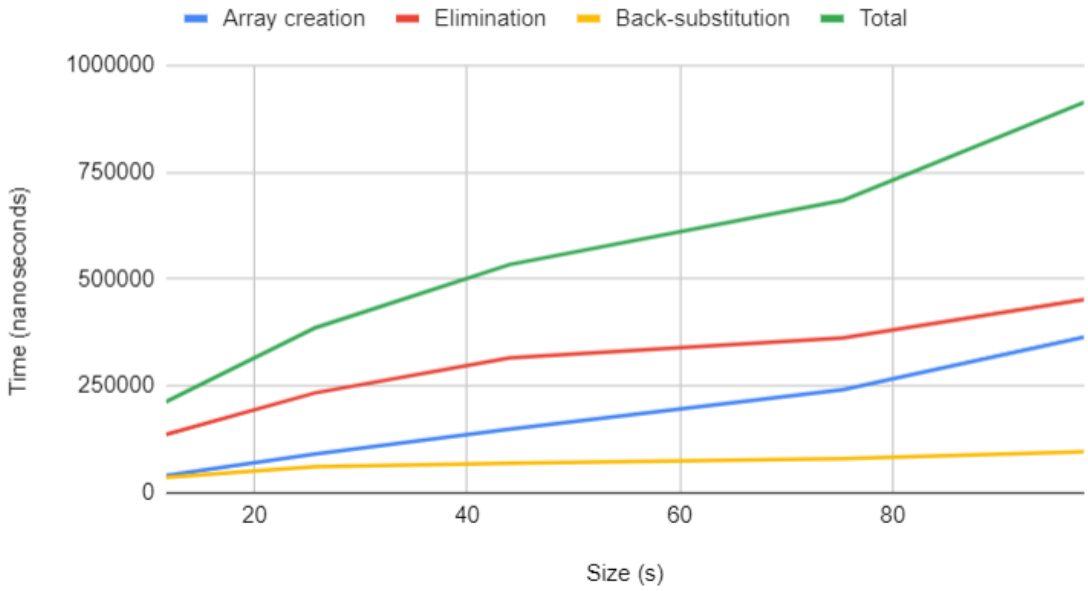

Figure 6. Performance of various steps of optimized GE for GF(2^6)

We see in Figure 6 that for smaller matrices elimination takes more time than the array creation. We observe that the array creation is the most time-consuming step in the algorithm for a sufficiently large system size. The back-substitution is the least time-consuming step therein. We ran decoding simulations with LDPC codes of length 2000, where after introducing the erasures, the number of unknowns s is taken in the range $10 < s < 100$. For the chosen systems of equations, we compare the execution time of the regular Gaussian elimination method and its optimized counterpart. The results are presented in Figure 7.

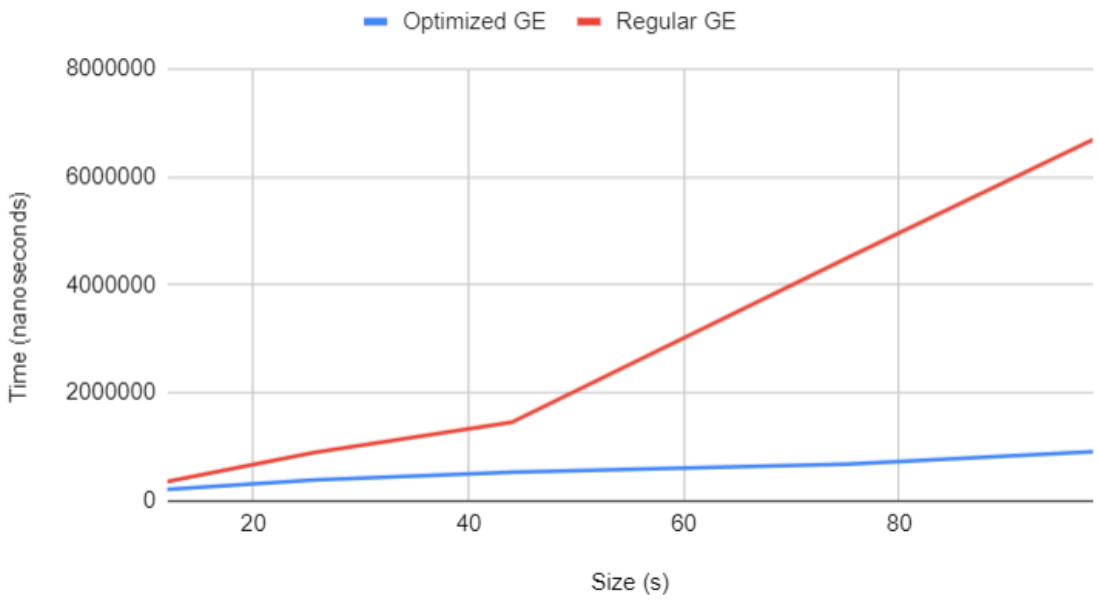

Figure 7. Performance difference between regular GE and optimized GE for GF(2^6)

From Figure 7 we can see that for s of size 100 an approximately 85% reduction in the runtime has been achieved by using the modified algorithm. Comparison of ML and BP decoding performance of rate $r = 1/2$ codes for two cases GF(2^4) and GF(2^6) are presented in the Appendix 1 and 2.

4.6 Implementation

This algorithm was implemented in Java and the source code can be found in the Github repository mentioned in Appendix 3. We present a pseudocode in two parts. Note that multiplication and division tables have been marked as protable and divtable respectively.

Algorithm 3: Optimized Gaussian elimination (part 1)

Input: matrix A, vector $\overline{b}$, table protable, table divtable

Output: solution vector $\overline{x}$

Initialize: table rows, table columns, vector $\overline{x}$.

l = number of equations;

s = number of unknowns;

for $i = 0$*;* $i < s$*;* i++ **do**

  for $j = 0$*;* $j < l$*;* j++ **do**

    if *A[i][j] is non-zero* **then**

      Add element $A[i][j]$ to row and column arrays

for $p = 0$*;* $p < s$*;* p++ **do**

  if *A[p][p] is zero* **then**

    for $i = 0$*;* $i <$ *columns[p] size;* i++ **do**

      if *p <= columns[p][i]* **then**

        switch i row with p row;

        break for-loop;

    update row and column arrays;

    for $i = 0$*;* $i <$ *columns[p] size;* i++ **do**

      α = divtable[A[indexrow][p]][A[p][p]];

      $\overline{b}$[columns[p][i]] = $\overline{b}$[columns[p][i]] $\oplus$ protable[alpha][$\overline{b}$[p]];

      for $j = 0$*;* $j <$ *rows[p] size;* j++ **do**

        A[columns[p][i]][rows[p][j]] =

        A[columns[p][i]][rows[p][j]] $\oplus$ protable[α][A[p][rows[p][j]]];

      update row and column arrays;

Algorithm 3: Optimized Gaussian elimination (part 2)

```
for i = 0; i < l; i++ do
    if if A[i] is a zero row then
        remove row i;
update row and column arrays;
check = false;
for i = s − 1; i ≥ 0; i– do
    sum = 0; for j = 0; j < rows[i] size; j++ do
        if check is true then
            if rows[i][j] > i then
                sum = sum ⊕ protable[A[i][rows[i][j]]][x̄[rows[i][j]]];
    check = true;
    if sum is zero then
        x̄[i] = divtable[b̄[i]][A[i][i]];
    else
        if sum == b̄[i] then
            x̄[i] = 0
        else
            x̄[i] = divtable[b̄[i] ⊕ sum][A[i][i]];
return x̄;
```

5 Conclusion

This thesis discussed the methods for fast solving of systems of linear equations which arise in LDPC decoding. Gaussian elimination algorithm was chosen to be optimized. The optimized Gaussian elimination was implemented in Java and also presented as a pseudocode.

The thesis surveyed belief propagation decoding over erasure channel and discussed why solving systems of equations is important in the runtime of the decoding process. Various methods of solving systems of linear equations were discussed.

The optimized Gaussian elimination algorithm was introduced and measured for systems of length $10 < s < 100$. The empirical results show that near quadratic time complexity was achieved, compared to the cubic complexity of the regular Gaussian elimination. This vastly speeds up the runtime of BP LDPC decoding algorithms.

Further research includes optimization of the process of creating and updating the column and row tables (which are now the most time consuming operations). Future research can also be done into iterative ways of solving system of equations and doing parallel computing to reach even better performance.

References

[1] Grcar, J. (2011). *How ordinary elimination became Gaussian elimination*. Historia Mathematica Volume 38, Issue 2, pp. 163-218.

[2] Casillo, N.G. *The History of Matrices and Modern Applications* SUNY Oswego, n.d. Retrived 01.12.2020.
`https://www.oswego.edu/writing-across-the-curriculum/sites/www.oswego.edu.writing-across-the-curriculum/files/mat_casillo_3-23-16.pdf`

[3] Le Gall, F. (2014). *Powers of Tensors and Fast Matrix Multiplication.* In Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, pg. 296–303.

[4] Landesman, A. *Notes on finite fields.* Stanford University. Retrived 28.03.2020.
`https://web.stanford.edu/~aaronlan/assets/finite-fields.pdf`

[5] Roth, R. (2006). *Introduction to Coding Theory. Linear block codes.* Cambridge University Press. pp.26-31, 56-59 and 15-16.

[6] Peterson, W. W., Weldon Jr., E. J. (1961). *Error-Correcting Codes.* MIT. press

[7] Shannon, C. E. (1948). *A mathematical theory of communication.* IThe Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, July 1948.

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

[9] Bernhard, M. J. Leiner (2005). *LDPC Codes – a brief Tutorial.*

[10] Lentmaier, M., Mitchell D. G. M., Fettweis G. P., Costello, D. J. (2010). *Asymptotically regular LDPC codes with linear distance growth and thresholds close to capacity.* 2010 Information Theory and Applications Workshop (ITA), San Diego, CA, pp. 1-8.

[11] Tanner, R. (1981). *A recursive approach to low complexity codes.* IEEE Transactions on Information Theory, vol. 27, no. 5, pp. 533-547, September 1981.

[12] El Hassani, S., Hamon, M. and Pénard, P. (2010). *A comparison study of binary and non-binary LDPC codes decoding.* SoftCOM 2010, 18th International Conference on Software, Telecommunications and Computer Networks, Split, Dubrovnik, pp. 355-359.

[13] Y. Fang, J. Zhang, L. Wang, and F. Lau. (2010). *BP-Maxwell decoding algorithm for LDPC codes over AWGN channels.* 6th Int. Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), pp. 1–4.

[14] Bocharova, I., Kudryashov, B., Lyamin, N., Frick, E., Rabi, M., Vinel, A. (2019). *Low Delay Inter-Packet Coding.* Vehicular Networks. Future Internet, 11(10), 212.

[15] Olmos, P. M., Mitchell D. G. M., and Costello, D. J. (2015) *OAnalyzing the finite-length performance of generalized LDPC codes.* Proc. IEEE Int. Symp. Inf. Theory (ISIT), pp. 2683–2687.

[16] Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., Spielman, D. A. and Stemann, V. (1997). *Practical loss-resilient codes.*Proc. ACM Symp. on Theory of Computing, pp. 150–159.

[17] Cramer, G. (1750). *Introduction à l'Analyse des lignes Courbes algébriques.* Geneva: Europeana. pp. 656–659.

[18] Cholesky, A. (1910). *Sur la résolution numérique des systèmes d'équations linéaires.* 2. December.

[19] Banachiewicz, T. (1937). *Zum Berechung der Determinanten, wie auch der Inversen, and zur daraut basierten Auflosung der systeme linearer Gleichungen.* Acta Astronomica, Ser. C.3, pp. 41–67.

[20] Turing, A. M. (1948). *Rounding-off errors in matrix processes.* The Quarterly Journal of Mechanics and Applied Mathematics, Volume 1, Issue 1, pp. 287–308.

[21] Francis, J. G. F. (1961) *The QR Transformation A Unitary Analogue to the LR Transformation — Part 1.* The Computer Journal, Volume 4, Issue 3, pp. 265–271.

[22] Francis, J. G. F. (1962). *The QR Transformation — Part 2.* The Computer Journal, Volume 4, Issue 4, pp. 332–345.

[23] Kublanovskaya V. N. (1962). *On some algorithms for the solution of the complete eigenvalue problem.* USSR Computational Mathematics and Mathematical Physics, Volume 1, Issue 3, pp. 637-657.

[24] Golub, G. H., Reinsch, C. (1970). *Singular value decomposition and least squares solutions.* Numerische Mathematik. 14 (5): pp. 403–420.

[25] Gauss, C. F. (1809) *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientum.*

[26] Legende, A. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes.*

[27] C.G.J. Jacobi (1845). *Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadraten vorkommenden lineare Gleichungen.* Astr. Nachr. , 22 : 523 (1845) pp. 297–306.

[28] Seidel, L. (1874). Abh. Bayer. Akad. Wiss. Math.-Naturwiss. Kl. , 11 : 3. pp. 81–108.

[29] Richardson, L.F. (1910). *The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam.* Philosophical Transactions of the Royal Society A. 210: pp. 307–357.

[30] Temple, G. (1939). *The General Theory of Relaxation Methods Applied to Linear Systems.* Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, 169(939), pp. 476-500.

[31] Young, D. M. Jr. (1954). *Iterative methods for solving partial difference equations of elliptical type.* PhD thesis, Harvard University.

[32] Frankel, S. P. (1950). *Convergence rates of iterative treatments of partial differential equations.* Math. Tables Aids Comput.4, pp. 65–75.

[33] Hestenes, M. R., Stiefel, E. (1952). *Methods of Conjugate Gradients for Solving Linear Systems*. Journal of Research of the National Bureau of Standards. 49 (6): p. 409.

[34] Moore, E. H. (1920). *On the reciprocal of the general algebraic matrix.* Bulletin of the American Mathematical Society. 26 (9): pp. 394–95.

[35] Penrose, R. (1955). *A generalized inverse for matrices.* Proceedings of the Cambridge Philosophical Society. 51 (3): pp. 406–13.

[36] James, M. (1978). *The Generalised Inverse.* The Mathematical Gazette, 62(420), pp. 109-114.

[37] Newton, I. (1711). *De analysi per aequationes numero terminorum infinitas.*

[38] Strassen, V. (1969). *Gaussian elimination is not optimal.* Numer. Math. 13, pp. 354-356.

[39] Cormen, T. H., Leiserson, C. E., Rivest R. L., Stein, C. (2009) *Introduction to Algorithms, 3rd ed.*. MIT Press, Cambridge, MA. pp. 828-831.

[40] Pan, V. Ya. (1978) *Strassen's algorithm is not optimal: Trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations.* Proceedings of the 19th Annual Symposium on Foundations of Computer Science, IEEE, pp. 166-176.

[41] Pan, V. Ya. (1978). *New fast algorithms for matrix operations.* IBM T. J. Watson Research Center Reporf RC-7555, (February 1979) and SIAM J. Contour. 9(2), (1980) Proceedings of the 19th Annual Symposium on Foundations of Computer Science, IEEE, pp. 166-176.

[42] Bini, D., Capovani, M., Romani, F., and Lotti, G. (1979). $O(n^2.7799)$ *complexity for* $n \times n$ *approximate matrix multiplication.* Information Processing Letters 8, 5 (1979), pp. 234–235.

[43] Schönhage, A. (1981). *Partial and Total Matrix Multiplication* SIAM Journal on Computing 10, 3, pp. 434–455.

[44] Schönhage, A. (1981). *New combinations of methods for the acceleration of matrix multiplication* Computer and Mathematics with Applications, pp. 73–125.

[45] Romani, F. (1982). *Some properties of disjoint sums of tensors related to matrix multiplication.* SIAM Journal on Computing 11, 2, pp. 263–267.

[46] Coppersmith, D. and Winograd, S. (1982). *On the asymptotic complexity of matrix multiplication.* SIAM Journal on Computing 11, 3, pp. 472–492.

[47] Strassen V. (1986). *The asymptotic spectrum of tensors and the exponent of matrix multiplication.* Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pp. 49– 54.

[48] Coppersmith, D. and Winograd, S. (1987). *Matrix Multiplication via Arithmetic Progressions* Journal of Symbolic Computation 9, 3 , pg. 251–280.

[49] Stothers, A. (2010) *On the Complexity of Matrix Multiplication.* PhD thesis, University of Edinburgh.

[50] Williams, V. (2012). *Multiplying matrices faster than Coppersmith-Winograd.* In Proceedings of the 44th Symposium on Theory of Computing, pp. 887–898.

[51] Irturk, A., Benson, B., Mirzaei, S., Kastner R. (2010). *GUSTO: An Automatic Generation and Optimization Tool for Matrix Inversion Architectures.* ACM Trans. Embedded Comput. Syst.

[52] Davey, M.C. and MacKay, D.J.C. (1998). *Low-density parity check codes over GF(q)*. IEEE Communications Letters 2(6), pp. 165-167.

Appendices

I. BP and ML LDPC decoding plot over GF(2^4)

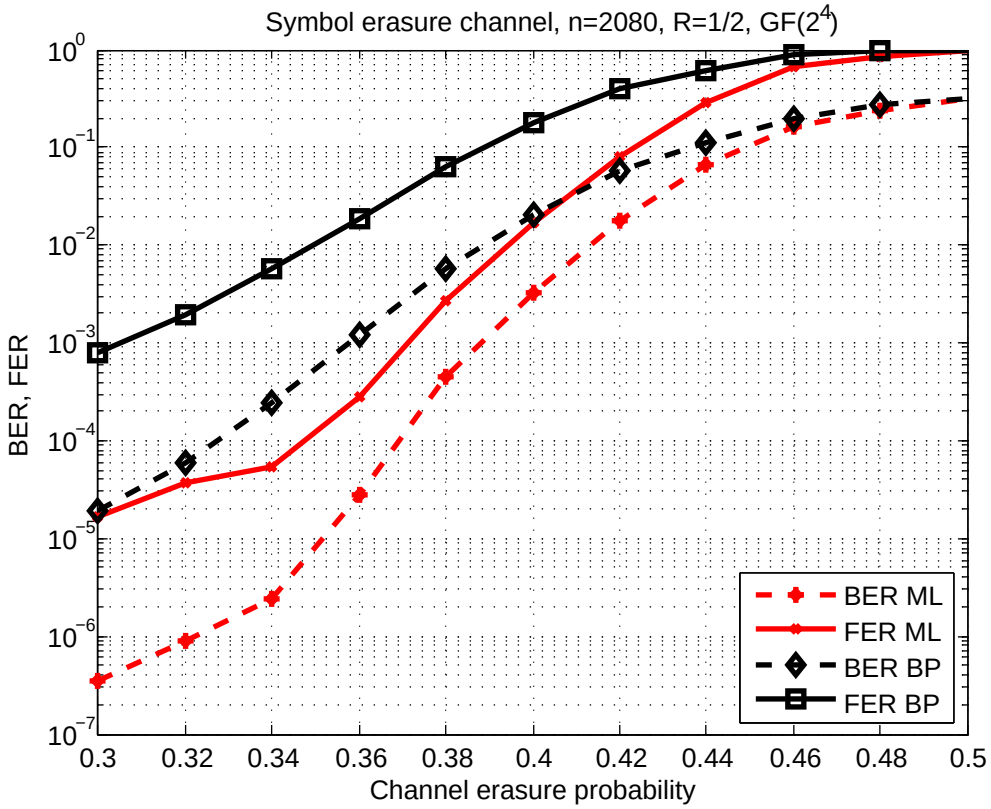

Figure 8. Simulation of BER and FER performance for LDPC codes with $n = 2080$, $R = 1/2$, over GF(2^4), for BP decoding and ML decoding with the improved Gaussian elimination.

II. BP and ML LDPC decoding plot over GF(2^6)

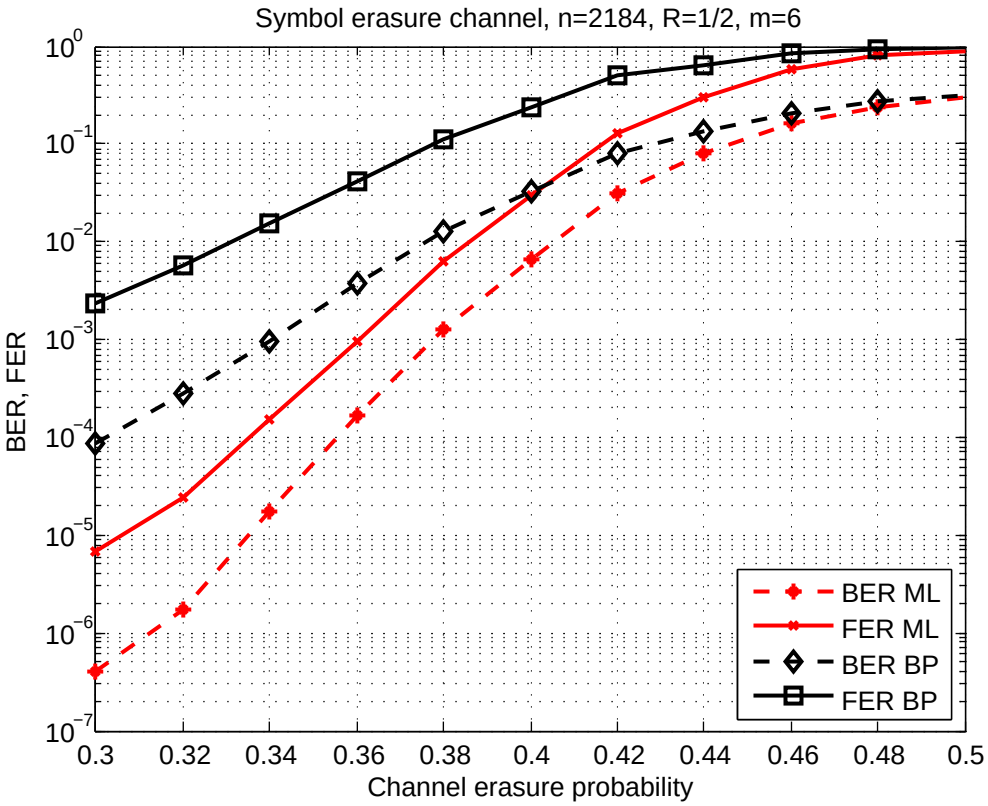

Figure 9. Simulation of BER and FER performance for LDPC codes with $n = 2184$, $R = 1/2$, over GF(2^6), for BP decoding and ML decoding with the improved Gaussian elimination.

III. Proposed algorithm source code

The optimized algorithm was implemented in Java programming language. It was uploaded to Github and the instructions on how to run the program are found there. Link to Github repository: `https://github.com/ukangur/LDPCGaussian`

IV. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Uku Kangur**,
*(*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,

   Fast solving of systems of linear equations in decoding of LDPC codes,
   *(*title of thesis)

   supervised by Vitaly Skachek, Boris Kudryashov 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.

Uku Kangur
07/05/2020