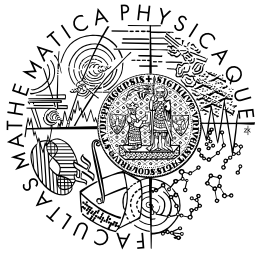

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Dalibor Procházka

Decoding of Reed-Solomon Codes

Department of Algebra

Supervisor of the bachelor thesis: doc. Mgr. et Mgr. Jan Žemlička, Ph.D.

Study programme: Mathematics for IT

Study branch: Mathematics

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor doc. Jan Žemlička for his time, patience and guidance in writing this thesis.

Title: Decoding of Reed-Solomon Codes

Author: Dalibor Procházka

Department: Department of Algebra

Supervisor: doc. Mgr. et Mgr. Jan Žemlička, Ph.D., Department of Algebra

Abstract: Reed-Solomon codes are a typical example of MDS codes, that are frequently used in practise. In this thesis, we go over three different algorithms of decoding these codes, including the initial view from the original article, as well as the modern approach of currently used algorithm and of another possible efficient algorithm. We compile various sources and unite them under the same notation. We describe in detail the theory behind each algorithm, show its correctness, discuss every algorithm's time complexity and demonstrate its steps on simple examples.

Keywords: Reed-Solomon codes, decoding, algorithms

Contents

Introduction

Reed-Solomon (RS) codes are a group of linear cyclic self-correcting codes. They were first introduced in 1960 as what we now call the traditional RS codes. The definition has been generalised, so when we talk about these codes nowadays, we usually refer to Generalised Reed-Solomon (GRS) codes, which are a broader family of codes, but with practically same properties as the original codes.
GRS are commonly used to this day. The most important practical uses are in encoding two dimensional bar codes, such as QR codes and PDF-417. Other important, though currently already a bit outdated usage is in data storage, specifically in encoding of CDs. Therefore it has great value and impact to study these codes properly and to find and describe as effective decoding algorithm as possible.
In this thesis, we take an in-depth look at a couple of these algorithms, starting with the original algorithm proposed by the first article, continuing with the efficient algorithm, which is nowadays most frequently used in practice, and ending with a new algorithm, which shows another efficient method of decoding these codes. We will describe the theory which is behind each of these algorithms, show the correctness of them and demonstrate their step by step process on simple examples.

1. Terms and Notations

In this chapter we will define terms and algorithms that we will be using throughout the whole text.

1.1 Codes and Linear Codes

We will first start by defining a reminding basic definitions from the self-correcting codes theory. Let $\mathbb{F}_q$ be a finite field o q elements where q is a prime power. We consider a vector $(v_0, v_1, \ldots, v_{n-1}) \in \mathbb{F}_q^n$ of length n, and we will call this vector a *word*. A *code* C is a set of these words, and the integer n is called the *length* of a code. If these words form a subspace of a linear vector space, then the code is a *linear code*. The *dimension* of a linear code is the dimension of the vector subspace, and we will denote it as k.
For two words $u, v \in C$, we define their *distance* $d(u, v)$ as $d(u, v) = |\{i \,|\, u_i \neq v_i\}|$. We define the *distance* d of a code C as

$$d = d(C) = \min(\{d(u, v) \mid u, v \in C, u \neq v\}).$$

Therefore we get three parameters characterizing a linear code. These parameters generally satisfy bounds $1 \leq k < n \leq q$ and $d \leq n - k + 1$. Using these parameters, we denote $[n, k, d]$ a linear code C of corresponding length, dimension and distance.
A matrix $G \in \mathbb{F}_q^{k \times n}$ is said to be the *generator matrix* of a linear code C, if its rows form a basis of C. A matrix $H \in \mathbb{F}_q^{(n-k) \times n}$ is the *parity-check matrix*, if $C = \ker H$.

1.2 Reed-Solomon Codes

Now its finally time to define the codes that we will be focusing on in this thesis. *General Reed-Solomon* (GRS) code $[n, k, d]$ over the field $\mathbb{F}_q$ is defined as $\ker H_{GRS}$, where

$$H_{GRS} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_0 & \alpha_1 & \ldots & \alpha_{n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_0^{n-k-1} & \alpha_1^{n-k-1} & \ldots & \alpha_{n-1}^{n-k-1} \end{pmatrix} \begin{pmatrix} v_0 & 0 & \ldots & 0 \\ 0 & v_1 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & v_{n-1} \end{pmatrix},$$

$\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{F}_q^*$ being distinct elements and $v_0, \ldots, v_{n-1} \in \mathbb{F}_q^*$.
Reed-Solomon code (RS) code $[n, k, d]$ over the field $\mathbb{F}_q$ is a special version of GRS code, for which $\exists \alpha \in \mathbb{F}_q^*$ of order n and $\exists b \in \mathbb{N}, 1 \leq b \leq n - 1$ such that $\alpha_i = \alpha^i$ and $v_i = \alpha^{bi}$ for $i = 0, \ldots, n - 1$. The parity-check matrix H_{RS} can be then expressed in a simpler way as

$$H_{RS} = \begin{pmatrix} 1 & \alpha^b & \ldots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \ldots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha^{b+n-k+1} & \ldots & \alpha^{(n-1)(b+n-k+1)} \end{pmatrix}.$$

This is the current definition of GRS codes, but these codes were originally defined through evaluating polynomials in given elements of a field. We will describe this procedure later in this chapter and see that there is a correspondence with the generator matrix.
GRS codes have many important properties. They are linear (as they are defined by the parity-check matrix) and they are also maximum distance separable (MDS). This means the distance of the code d satisfies the equation $d = n-k+1$ and is therefore the biggest distance possible for the given code parameters (as all codes satisfy the bound $d \leq n - k + 1$).
All these properties of codes can be found in Roth [2006].

1.3 Coding Process

Our main focus will be on decoding algorithms, but we first describe the structure of the whole coding process. We fix the following notation for words. A *message* $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ is the message word before encoding. A *codeword* $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$, is the word we get by encoding the message (as will be described below). These words are the elements which are forming the given code. A *received word* $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1}), \in \mathbb{F}_q^n$ is the word received by transmitting the codeword. The received word is the codeword changed by adding an *error word* $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1}), \in \mathbb{F}_q^n$ to it. We say a *transmission error* occurred at position $i = 0, 1, \ldots, n-1$ if the i-th position of the received word $\boldsymbol{b}$ is different from i-th position of the sent codeword $\boldsymbol{c}$: $b_i \neq c_i$, or equivalently the i-th position of the error word $\boldsymbol{e}$ is nonzero: $e_i \neq 0$. We denote T the set of error locations, meaning $T = \{i \mid e_i \neq 0, i = 0, \ldots, n-1\}$.
The whole idea of coding is adding some redundancy to our message, so we can retrieve the original message even if some number of errors occurred during the transmission. The process consists of encoding a message into a corresponding codeword, then transmitting it through the channel. From mathematical point of view, the channel gives us probabilities of errors happening in certain positions, it does not deal with the actual transmission. As we said, the codeword can be changed by the error word, and our goal is to correctly tell the original codeword from the received word. The process can be demonstrated using this simple scheme:

$$\mathbf{m} \xrightarrow{encoding} \mathbf{c} \xrightarrow{transmission:+\mathbf{e}} \mathbf{b}.$$

Note that it is not necessary to retrieve the message right away, it is sufficient just to get the codeword or the error word, as we require the coding to be injective, therefore for each codeword, there is only one unambiguous corresponding message, and the codeword can be easily obtained from knowing the received and the error word.
Important result of the coding theory is that we are able to correct t errors (meaning t nonzero positions of the error word), where $t < d/2$. So, in the case of MDS codes such are GRS codes, we can correct $t < \frac{n-k+1}{2}$ errors. Therefore all algorithms, which we will describe, have the maximum number of errors, given by this bound, as an assumption in order to work correctly.

1.4 Encoding

We now describe the process of encoding a message into a codeword by evaluation of polynomial. For this process we fix n different elements $a_0, a_1, \ldots, a_{n-1}$ of $\mathbb{F}_q$. When we want to encode a message of k elements $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}_q^k$, we first consider the *message polynomial*

$$f = m_0 + m_1 x + \ldots + m_{k-1} x^{k-1},$$

then we define the corresponding codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ by computing c_i as

$$c_i = f(a_i) \in \mathbb{F}_q, \;\; i = 0, 1, \ldots, n-1.$$

The coding can be interpreted as a mapping

$$\begin{gathered} E : \mathbb{F}_q^k \to \mathbb{F}_q^n \\ (m_0, m_1, \ldots, m_{k-1}) \mapsto (f(a_0), f(a_1), \ldots, f(a_{n-1})) \end{gathered}$$

where $m_0, m_1, \ldots, m_{k-1}$ are coefficients of the message polynomial f of degree $\leq k-1$

$$f = m_0 + m_1 x^1 + \ldots + m_{k-1} x^{k-1}, m_i \in \mathbb{F}_q, k < q.$$

Note that described encoding is non-systematic, which means that the original message $\mathbf{m}$ is not a part of the codeword $\mathbf{c}$.
The evaluation of the message polynomial in the element $a \in \mathbb{F}_q$ can be interpreted as the dot product of vectors $(1, a, a^2, \ldots, a^{k-1})$ and $(m_0, m_1, \ldots, m_{k-1})$. When we represent it in a matrix form, we get an expression for the mapping E

$$E(\boldsymbol{m}) = \begin{pmatrix} 1 & a_0 & \ldots & a_0^{k-1} \\ 1 & a_1 & \ldots & a_1^{(k-1)} \\ \vdots & \vdots & & \vdots \\ 1 & a_{n-1} & \ldots & a_{n-1}^{(k-1)} \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ \vdots \\ m_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}.$$

This is the general idea of linear encoding, when we associate a message with its codeword by multiplying said message with a given matrix. This idea of evaluating polynomials is a typical construction of MDS codes, and the resulting code has the same properties as the one defined by the parity-check matrix. The only important difference is that when evaluating polynomials we allow zero to be one of the elements, whereas the elements of the parity-check matrix must be nonzero. This has a good reason, as the algorithm most frequently used in practise nowadays needs inverses of these elements.
Furthermore, if we look at the submatrix given by k fixed different elements $a_i \in \mathbb{F}_q, i = 0, 1, \ldots, k-1$, we get a matrix whose coefficient determinant for variables $(m_0, m_1, \ldots, m_{k-1})$ is

$$\begin{vmatrix} 1 & a_0 & \ldots & a_0^{k-1} \\ 1 & a_1 & \ldots & a_1^{(k-1)} \\ \vdots & \vdots & & \vdots \\ 1 & a_{k-1} & \ldots & a_{k-1}^{(k-1)} \end{vmatrix}$$

This is Vandermonde determinant, as all elements a_i are different. Its value is $\prod_{i>j}(a_i - a_j) \neq 0$ (as can be found in Barto and Tůma [2023]), therefore the matrix is regular and the system of linear equations with these coefficients has one solution. This will be important in the chapter about the RS 1960 algorithm.

1.5 Extended Euclidean Algorithm

We will now define notations and show couple useful properties of the well-known Extended Euclidean Algorithm (EEA) for polynomials. This algorithm will be needed many times for proofs of correctness of decoding algorithms in later chapters. EEA ends after $m+1$ steps with $r_{m+1} = 0$ and $GCD(r_0, r_1) = r_m$. The degrees of r_i decrease strictly, as the division remainder's degree is always lower than the divisor's degree, and each steps satisfies relation $r_i = u_i r_0 + v_i r_1$.
Lemma 1.1 shows a few more useful properties of EEA.

Algorithm 1: Extended Euclidean Algorithm for Polynomials

Input : nonzero polynomials $r_0, r_1 \in \mathbb{F}_q[x], \deg r_0 \geq \deg r_1$
Output: polynomials r_m, u_m, v_m satisfying $r_m = u_m r_0 + v_m r_1$

$(u_0, u_1) = (1, 0), (v_0, v_1) = (0, 1), i = 1$
while $r_i \neq 0$ **do**
  $q_i = r_{i-1}$ div r_i
  $r_{i+1} = r_{i-1}$ mod r_i
  $(u_{i+1}, v_{i+1}) = (u_{i-1} - q_i u_i, v_{i-1} - q_i v_i)$
  $i = i + 1$;
end
return (r_i, u_i, v_i);

Lemma 1.1. *Using the notation of the EEA the following conditions hold:*

1. $\deg v_{i+1} + \deg r_i = \deg r_0$ *for* $i = 0, \ldots, m$,
2. $\deg u_{i+1} + \deg r_i = \deg r_1$ *for* $i = 1, \ldots, m$,
3. $\deg v_i \leq \deg r_0$ *for* $i = 0, \ldots, m$,
4. $\deg u_i \leq \deg r_1$ *for* $i = 0, \ldots, m$.

Proof. 1. By induction on i. For $i = 0$ we have

$$\deg v_1 + \deg r_0 = \deg 1 + \deg r_0 = \deg r_0,$$

and for $i = 1$ we get, using the relation $q_i = r_{i-1}$ div r_i,

$$\begin{aligned}\deg v_2 + \deg r_1 &= \deg(v_0 - q_1 v_1) + \deg r_1 = \deg q_1 + \deg r_1 \\ &= \deg r_0 - \deg r_1 + \deg r_1 = \deg r_0.\end{aligned}$$

Now suppose that the equality holds for $i \geq 1$. Then for $i+1$ we get

$$\deg v_{i+2} + \deg r_{i+1} = \deg(v_i - q_{i+1} v_{i+1}) + \deg r_{i+1}.$$

We now use the induction step on $\deg v_i$ and $\deg(q_{i+1}v_{i+1})$. Firstly

$$\deg v_i = \deg r_0 - \deg r_{i-1},$$

and secondly, using again the relation $q_i = r_{i-1} \operatorname{div} r_i$,

$$\begin{aligned}\deg(q_{i+1}v_{i+1}) &= \deg q_{i+1} + \deg v_{i+1} = \deg r_i - \deg r_{i+1} + \deg r_0 - \deg r_i \\ &= \deg r_0 - \deg r_{i+1}\end{aligned}$$

As the degrees of r_i decrease strictly, we have

$$\deg v_i = \deg r_0 - \deg r_{i-1} < \deg r_0 - \deg r_{i+1} = \deg(q_{i+1}v_{i+1}).$$

Therefore

$$\deg(v_i - q_{i+1}v_{i+1}) = \deg(q_{i+1}v_{i+1}) = \deg r_0 - \deg r_{i+1},$$

which implies

$$\deg v_{i+2} + \deg r_{i+1} = \deg r_0 - \deg r_{i+1} + \deg r_{i+1} = \deg r_0.$$

2. By induction on i, similarly to 1. For $i = 1$ we have

$$\deg u_2 + \deg r_1 = \deg(u_0 - q_1 u_1) + \deg r_1 = \deg(1 - q_1 \cdot 0) + \deg r_1 = \deg r_1.$$

For $i + 1, i \geq 1$, we get

$$\begin{aligned}\deg u_{i+2} + \deg r_{i+1} &= \deg(u_i - q_{i+1}u_{i+1}) + \deg r_{i+1} = \deg(q_{i+1}u_{i+1}) \\ &= \deg r_{i+1} = \deg r_1 - \deg r_{i+1} + \deg r_{i+1} = \deg r_1,\end{aligned}$$

using the induction hypothesis and argumentation exactly as in 1. Note that this also holds for the edge case of $i = 1$, because

$$\deg u_1 = \deg 0 < \deg(q_2 u_2) = \deg(q_2(u_0 - q_1 u_1)) = \deg q_2,$$

as $(u_0, u_1) = (1, 0)$.
3. Is implied by equality 1., as for $i = 0$ holds

$$\deg v_0 = \deg 0 \leq \deg r_0,$$

and for $i \geq 1$, we obtain

$$\deg v_i = \deg r_0 - \deg r_{i-1} \leq \deg r_0.$$

4. Is implied by equality 2., as for $i = 0$ and $i = 1$ respectively holds

$$\deg u_0 = \deg 1 \leq \deg r_1, \qquad \deg u_1 = \deg 0 \leq \deg r_1,$$

and for $i \geq 2$ we get

$$\deg u_i = \deg r_1 - \deg r_{i-1} \leq \deg r_1.$$

□

1.6 Partial Euclidean Algorithm

In two of the decoding algorithms, we will need a modified version of the Euclidean Algorithm for polynomials. It differs from the classical one in the condition which stops the algorithm: in this version, the algorithm stops the first time the degree of the remainder is lower than a given threshold. We will denote this algorithm as the Partial Euclidean Algorithm (PEA) for polynomials.

Algorithm 2: Partial Euclidean Algorithm for Polynomials

Input : nonzero polynomials $r_0, r_1 \in \mathbb{F}_q[x], \deg r_0 \geq \deg r_1$, threshold t
Output: polynomials r_m, u_m, v_m satisfying $r_m = u_m r_0 + v_m r_1$, $\deg\, r_m < t$

$(u_0, u_1) = (1, 0), (v_0, v_1) = (0, 1), i = 1$
while $\deg r_i \geq t$ **do**
    $q_i = r_{i-1} \text{ div } r_i$
    $r_{i+1} = r_{i-1} \bmod r_i$
    $(u_{i+1}, v_{i+1}) = (u_{i-1} - q_i u_i, v_{i-1} - q_i v_i)$
    $i = i + 1$;
end
return (r_i, u_i, v_i);

2. RS 1960

In this chapter we will work with the finite field $\mathbb{F}_{2^N}$, so for our initial notation holds $q = 2^N$, where N is an integer. To represent this field we choose a generator α of its cyclic group $\mathbb{F}_{2^N}^* = \langle\alpha\rangle$. The field is then represented as $\mathbb{F}_2(\alpha) \cong \mathbb{F}_2[x]/(p)$, where p is a suitable irreducible polynomial of degree N over $\mathbb{F}_2[x]$ and α is its root. We will be using all the elements of the field to evaluate the message polynomial when encoding a message, meaning $n = 2^N$.
In this chapter, we draw from Reed and Solomon [1960].

2.1 Decoding

After receiving a received word $\boldsymbol{b}$, we can decode the original message $\boldsymbol{m}$ by solving any system of k linear equations out of 2^N received (supposing no errors occurred during the transmission):

$$\begin{aligned} f(0) &= m_0 \\ f(\alpha) &= m_0 + m_1\alpha^1 + \ldots + m_{k-1}\alpha^{k-1} \\ f(\alpha^2) &= m_0 + m_1\alpha^2 + \ldots + m_{k-1}\alpha^{2(k-1)} \\ &\vdots \\ f(1) &= m_0 + m_1 + \ldots + m_{k-1} \end{aligned}$$

Every k of these equations are linearly independent as the coefficient determinant for, for instance, equations $f(\alpha), \ldots, f(\alpha^k)$ is

$$\begin{vmatrix} 1 & \alpha & \ldots & \alpha^{k-1} \\ 1 & \alpha^2 & \ldots & \alpha^{2(k-1)} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha^k & \ldots & \alpha^{k(k-1)} \end{vmatrix}$$

which is Vandermonde determinant whose value is $\prod_{i>j}(\alpha^i - \alpha^j) \neq 0$ in the case of α^i being all different. This is true here as we consider the powers $\alpha^i, i = 1, \ldots, k < 2^N$ and α is the generator of $\mathbb{F}_{2^N}^*$. Therefore we get $\binom{2^N}{k}$ up to ordering different systems of k different linear equations to calculate the original message $(m_0, m_1, \ldots, m_{k-1})$.

Our goal however is to decode received words that also include some errors. For that we define the following notation. By the term *determination* of a k-tuple $(r_0, r_1, \ldots, r_{k-1})$ we will understand a system of k independent linear equations in the form $f(\alpha^i) = x_0 + x_1\alpha^i + \ldots + x_{k-1}\alpha^{i(k-1)}$ whose solution is $(r_0, r_1, \ldots, r_{k-1})$. A determination is *wrong* when its solution does not equal the sent message, otherwise the determination is *correct*.

Lemma 2.1. *For t transmission errors we can get a maximum of $\binom{t+k-1}{k}$ wrong determinations for a k-tuple (considering a different k-tuple from the original message).*

Proof. Every k different linear equations define a determination for some k-tuple $r = (r_0, r_1, \ldots, r_{k-1})$. If no errors occurred in the positions determined by the chosen equations, the determination is correct, as the only solution to the system of k unchanged linear equations is the original message. If at least one of the equations was changed by a transmission error, we do get different solution from the original message, so this system of equations would be a wrong determination. So in order to obtain a wrong determination, we have to choose at least one out of t equations changed by the errors and supplement the system of equations by up to $k-1$ correct equations. Therefore every wrong determination consists of i changed equations and $k-i$ unchanged equations, $i = 1, \ldots, \min(t, k)$. Furthermore, we can choose from only $k-1$ unchanged equations as having the option to choose k unchanged equations would give us the correct solution and not the k-tuple which we are counting the wrong determinations for. Therefore the total number of wrong determinations of a single k-tuple is at most $\binom{t+k-1}{k}$. □

This lemma implies a voting algorithm to determine the correct solution, which we will describe shortly. By a *vote* for a word $r = (r_0, r_1, \ldots, r_{k-1}) \in \mathbb{F}_{2^N}^k$ we mean a unique system of k linear equations whose solution is r. We will denote V_r the number of votes received by the word r.

Algorithm 3: Voting Decoding Algorithm

Input : received word $\mathbf{b} = (b_0, b_1, \ldots, b_{2^N-1})$, maximum number of errors occurred t

Output: sent message $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$

```
Max = 0
while ∃ unused combination of k equations and Max ≤ C(t+k-1, k) do
    compute solution r of an unused combination of k equations (as
      described above)
    V_r = V_r + 1
    Max = max(Max, V_r);
end
if Max > C(t+k-1, k) then
    output r
else
    output "Decoding failure" as more than t errors occurred during the
      transmission
end
```

Note that we have total of $\binom{2^N-t}{k}$ correct determinations, which gives us total of $\binom{2^N}{k} - \binom{2^N-t}{k}$ wrong determinations. Therefore the algorithm works only in the case of the number of correct determinations being higher than the number of wrong determinations, that is

$$\binom{2^N - t}{k} > \binom{t + k - 1}{k},$$

which is equivalent to

$$2^N - t > t + k - 1,$$

which implies

$$t < \frac{2^N - k + 1}{2}.$$

Therefore the algorithm is not able to correct more than $\frac{2^N-k+1}{2}$ errors. We can correct up to $t = \frac{2^N-k-1}{2}$ errors for k odd and up to $t = \frac{2^N-k}{2}$ errors for k even, which corresponds to the expectation given by the distance $d = 2^N - k + 1$.

2.2 Comment about the Time Complexity of Algorithm 2

The algorithm computes solutions of k linear equations with k variables, each solution with the time complexity $O(k^3)$. However, in case of coding large amount of words using the same code, we can precompute inverses of the matrices used in the equation solving, making the process of finding the solution much quicker. As for the while cycle, it would need to be repeated up to $\binom{2^N}{k}$ times in the worst case scenario, which we can limit by

$$\binom{2^N}{k} \leq \left(\frac{e2^N}{k}\right)^k \leq 2^{Nk}$$

for $k \geq 3$.
It is also interesting to look at the case of no errors occurring during the transmission. The while cycle will run $\binom{t+k-1}{k} + 1$ times, as every determination is correct and will give us correct solution. We can limit this by

$$\binom{t+k-1}{k} + 1 < \binom{\frac{2^N-k+1}{2} + k - 1}{k} + 1 = \binom{\frac{2^N+k-1}{2}}{k} + 1.$$

These limitations do not give us very good idea on how many times will the while cycle run, we will demonstrate it better on the following example.

2.3 Example

Let $N = 3$ and $k = 3$ therefore the distance of this code is $d = 2^3 - 3 + 1 = 6$ and the code is able to correct up to $\frac{2^3-3-1}{2} = 2$ errors. The field $\mathbb{F}_8$ is represented as $\mathbb{F}_2(\alpha)$ where α is a root of the irreducible polynomial $p = x^3 + x + 1 \in \mathbb{F}_2[x]$, therefore

$$\begin{aligned}\mathbb{F}_8 = \mathbb{F}_2(\alpha) &= \{0, \alpha, \alpha^2, \alpha+1, \alpha^2+\alpha, \alpha^2+\alpha+1, \alpha^2+1, 1\} \\ &= \{0, \alpha, \alpha^2, \ldots, \alpha^7\} \cong \mathbb{F}_2[x]/(x^3+x+1).\end{aligned}$$

The message polynomial for general message $\mathbf{m} = (m_0, m_1, m_2) \in \mathbb{F}_8^3$ is

$$f = m_0 + m_1 x + m_2 x^2 \in \mathbb{F}_2[x].$$

Let us consider the message $\mathbf{m} = (\alpha, \alpha^2, \alpha^2 + \alpha + 1)$, which defines the message polynomial

$$f = \alpha + \alpha^2 x + (\alpha^2 + \alpha + 1)x^2.$$

To encode this message, we evaluate the polynomial f in all field's elements $0, \alpha, \ldots, 1$, getting the codeword $\mathbf{c} = (\alpha, 0, 0, \alpha+1, \alpha, 1, \alpha+1, 1)$.
Suppose that an error $\mathbf{e} = (\alpha, 1, 0, 0, 0, 0, 0, 0)$ occurred during the transmission, changing 2 symbols and causing the received word to be

$$\mathbf{b} = \mathbf{c} + \mathbf{e} = (0, 1, 0, \alpha+1, \alpha, 1, \alpha+1, 1).$$

To decode the message we keep computing solution for each of $\binom{8}{3} = 56$ systems of 3 linear equations $b_i = f(\alpha^i) = m_0 + m_1\alpha^i + m_2\alpha^{2i}$, $i = 0, 1, \ldots, 7$, until one of the solution receives more votes than the limit given by the maximum number of errors or we run out of equations. In our case, the algorithm is bale to correct up to 2 errors, so the limit of votes is $\binom{2+3-1}{3} = 4$. Therefore the algorithm stops once any of the solutions receives 5 votes. This solution will also be the one returned. In this case, the solution which reached the threshold is $(\alpha, \alpha^2, \alpha^2+\alpha+1)$. If we let the algorithm compute all out of $\binom{8}{3} = 56$ systems of equations, we would see that this solution received 20 votes total. We see that the decoded word is indeed the sent message $\mathbf{m}$ so the algorithm successfully corrected 2 errors.
If we suppose that no error happens during the transmission and every combination of equations give us the correct solution, we would need to run the while cycle less than $\binom{\frac{8+3-1}{2}}{3} + 1 = 11$ times as given by the general limitation earlier in this chapter. But we are able to calculate the exact number of while cycle rounds given by the number of votes needed to consider a word to be the correct one, which is only 5 in our case. So the algorithm runs only a small fraction of time compared to the total number of combinations, which is 56, if no errors occur.

2.4 Translation of $\mathbb{F}_{2^N}$ into Binary Representation

Consider an irreducible polynomial $p(x) \in \mathbb{F}_2[x]$ of degree N whose root generates $\mathbb{F}_{2^N}$ as described in the beginning of this section,

$$p = p_0 + p_1 x + \ldots + x^N, \quad p_i \in \mathbb{F}_2$$

Every field element a can be naturally represented as a binary sequence of N bits, as $\mathbb{F}_{2^N}$ is represented as division remainders of p, so i-th bit of the binary representation of a corresponds to the coefficient a_{i-1} at x^{i-1}:

$$a = a_0 + a_1 x + \ldots + a_{N-1}x^{N-1} \mapsto (a_0, a_1, \ldots, a_{N-1}), \quad a_i \in \mathbb{F}_2.$$

Therefore we can understand E as a mapping of sequences of kN bits into sequences of $2^N N$ bits.
Other way of representing the elements of F_{2^N}, which is also used in the original article and in the example, is with the use of recurrent linear sequences. We compute the sequence of elements of $\mathbb{F}_2$ $u_0, u_1, \ldots, u_N, u_{N+1}, u_{N+2}, \ldots$ using the equation given by the polynomial p:

$$u_{N+j} = p_0 u_{0+j} + p_1 u_{1+j} + \ldots + p_{N-1} u_{N-1+j}.$$

We get a sequence which depends on the initial choice of $(u_0, u_1, \ldots, u_{N-1})$. We then represent the elements of $\mathbb{F}_{2^N}$ as N-tuples and define multiplication by the generator of the field α as translation in the sequence, meaning

$$\alpha = (u_0, \ldots, u_{N-1}), \alpha^2 = (u_1, \ldots, u_N), \ldots.$$

In the following section we show that this sequence has a period long enough to cover all elements of the cyclic group of the field for suitable polynomial p.

2.4.1 Period of the Sequence

First, let us consider a general element of the field $\mathbb{F}_{2^N}$, which has the form of $\beta = a_{N-1}\alpha^{N-1} + \ldots + a_1\alpha + a_0$, where α is a generator of the field and $a_i \in \mathbb{F}_2, i = 0, \ldots, N-1$. The multiplication by α can be expressed as

$$\begin{aligned}\alpha\beta =& \alpha(a_{N-1}\alpha^{N-1} + \ldots + a_1\alpha + a_0) \mod p(\alpha)\\ =& a_{N-1}\alpha^N + a_{N-2}\alpha^{N-1} \ldots + a_1\alpha^2 + a_0\alpha \mod p(\alpha).\end{aligned}$$

Continuous multiplying by α gives us a sequence of elements of $\mathbb{F}_{2^N}$. Lets denote the elements of the sequence as $\beta_0 = \beta, \beta_1 = \alpha\beta, \ldots$, given by the relation $\beta_i = \alpha\beta_{i-1} = \alpha^i\beta_0 = \sum_{j=0}^{N-1} a_{i,j}\alpha^j$. This sequence has a period of length $2^N - 1$ as α is the generator and has the multiplicative order of $2^N - 1$.
We now define a new sequence $\{v_i\}_{i=0}^{\infty}$ of coefficients at α^{N-1} of the sequence $\{\beta_i\}_{i=0}^{\infty}$, meaning $v_i = a_{i,N-1}$. All elements of $\{v_i\}$ are determined by $\{\beta_i\}$, therefore $\{v_i\}$ also has the length of $2^N - 1$. We now show that the sequence $\{v_i\}$ is up to a shift the same as the sequence $\{u_i\}$ defined earlier by the polynomial p as $u_{N+j} = p_0u_{0+j} + p_1u_{1+j} + \ldots + p_{N-1}u_{N-1+j}$.
The operation of multiplying by the element α can be written as

$$\begin{aligned}\alpha\beta =& \alpha(a_{N-1}\alpha^{N-1} + \ldots + a_1\alpha + a_0) \mod p(\alpha)\\ =& \alpha\beta + a_{N-1}p(\alpha) = \alpha\left(\sum_{i=0}^{N-1} a_i\alpha^i\right) + a_{N-1}\left(\sum_{i=0}^{N} p_i\alpha^i\right)\\ =& a_{N-1}p_0 + \sum_{i=1}^{N-1}(a_{i-1} + a_{N-1}p_i)\alpha^i.\end{aligned}$$

This corresponds to counting $\mod p(\alpha)$, which in practice means that every member α^N can be substituted for $\alpha^N = \sum_{i=0}^{N-1} p_i\alpha^i$. We see that the absolute coefficient of the new element is p_0 times the coefficient at α^{N-1} of the previous element. All other coefficients are computed by increasing the exponent of their monomial by 1 and then adding the product of α^{N-1} and coefficient p_i corresponding to the new exponent.
Now suppose that we produced $l \geq N$ corresponding elements of both sequences $\{\beta_i\}$ and $\{v_i\}$, the last elements being $\beta_l = \sum_{j=0}^{N-1} a_{l,j}\alpha^j$ and v_l, where $v_l = a_{l,N-1}$. The next step produces $\beta_{l+1} = \sum_{j=0}^{N-1} a_{l+1,j}\alpha^j$ and v_{l+1}, where $v_{l+1} = a_{l+1,N-1}$ and also $a_{l+1,0} = a_{l,N-1}p_0 = v_lp_0$. Let us observe the evolution of this coefficient v_lp_0 through next steps. In the next step it will move up to monomial α as $a_{l+2,1} = v_lp_0 + v_{l+1}p_1$. Continuing this process up to element β_{l+N} and its monomial α^{N-1} we get

$$a_{l+N,N-1} = \sum_{i=0}^{N-1} v_{i+l}p_l = v_{N+l}.$$

This corresponds exactly to the relation of the sequence $\{u_i\}$, which means that both sequences $\{u_i\}$ and $\{v_i\}$ produce the same output up to ordering and both have the period length of $2^N - 1$. Therefore, we can associate elements of $\mathbb{F}_{2^N}$ with N-tuples of elements of the sequence u_i and multiplication by α is defined as translation in the sequence, meaning

$$\begin{aligned}
\alpha &\sim (u_0, \ldots, u_{N-1}) \\
\alpha^2 &\sim (u_1, \ldots, u_N) \\
&\vdots \\
\alpha^{2^N-1} &\sim (u_{2^N-2}, u_0, \ldots, u_{N-2}).
\end{aligned}$$

Depending on the initial choice of $\alpha \sim (u_0, \ldots, u_{N-1})$ we get one of $2^N - 1$ possible representations.

2.4.2 Example

Consider the same situation as in Example 2.3. To use binary representation of the encoding, we get the recurrent formula defined by the polynomial $p = x^3 + x + 1$

$$u_j = u_{j-2} + u_{j-3}.$$

Choosing the initial state $(u_0, u_1, u_2) = (1, 1, 0)$ we get the sequence

$$\{u_j\} = (1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, \ldots).$$

The sequence $\{u_j\}$ has period 7, meaning $u_7 = u_0$, $u_8 = u_1$, ... We therefore represent the field's elements as follows:

$$\begin{aligned}
0 &\sim (0, 0, 0) \\
\alpha &\sim (1, 1, 0) \\
\alpha^2 &\sim (1, 0, 0) \\
\alpha^3 &= \alpha + 1 \sim (0, 0, 1) \\
\alpha^4 &= \alpha^2 + \alpha \sim (0, 1, 0) \\
\alpha^5 &= \alpha^2 + \alpha + 1 \sim (1, 0, 1) \\
\alpha^6 &= \alpha^2 + 1 \sim (0, 1, 1) \\
\alpha^7 &= 1 \sim (1, 1, 1)
\end{aligned}$$

Using this representation on our message and codeword, we get

$$\boldsymbol{m} = (110, 100, 101) \mapsto \boldsymbol{c} = (110, 000, 000, 001, 110, 111, 001, 111).$$

3. Current Algorithm

In this chapter we will discuss an algorithm which is nowadays most commonly used in practice. We will be using Roth [2006] as our source for this chapter. Let us consider a code over a finite field $\mathbb{F}_q$, the code being defined by its parity-check matrix as Ker H_{GRS}, where

$$H_{GRS} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_0 & \alpha_1 & \dots & \alpha_{n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_0^{n-k-1} & \alpha_1^{n-k-1} & \dots & \alpha_{n-1}^{n-k-1} \end{pmatrix} \begin{pmatrix} v_0 & 0 & \dots & 0 \\ 0 & v_1 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & v_{n-1} \end{pmatrix},$$

$\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F}_q^*$ being distinct elements and $v_0, \dots, v_{n-1} \in \mathbb{F}_q^*$.
We supppose that $t \leq (d-1)/2$ transmission errors happened ($d = n - k + 1$ being the distance of the code). We denote the error word $\boldsymbol{e} = (e_0, e_1, \dots, e_{n-1})$ satisfying the relation $\boldsymbol{b} = \boldsymbol{c} + \boldsymbol{e}$, $\boldsymbol{c}$ being a codeword and $\boldsymbol{b}$ being the received word. We denote T the set of error locations, meaning $T = \{i \,|\, e_i \neq 0, i = 0, \dots, n-1\}$, therefore $|T| = t \leq (d-1)/2$.
We will now describe and analyze the steps of the algorithm.

3.1 Syndrome Computation

First step of the algorithm is the computation of the *syndrome*, which is for $\boldsymbol{b}$ and with respect to H_{GRS} defined as

$$\begin{pmatrix} S_0 \\ S_1 \\ \vdots \\ S_{d-2} \end{pmatrix} = H_{GRS}\boldsymbol{b}^\top.$$

Individual syndrome entries can be therefore expressed as

$$S_i = \sum_{j=0}^{n-1} b_j v_j \alpha_j^i, \qquad i = 0, \dots, d-2.$$

We then define the *syndrome polynomial* $S \in \mathbb{F}_q[x]$ as

$$S = \sum_{i=0}^{d-2} S_i x^i.$$

The syndrome of the received word $\boldsymbol{b}$ equals the syndrome of the error word $\boldsymbol{e}$, because

$$S = H_{GRS}\boldsymbol{b}^\top = H_{GRS}(\boldsymbol{c} + \boldsymbol{e})^\top = H_{GRS}\boldsymbol{c}^\top + H_{GRS}\boldsymbol{e}^\top = H_{GRS}\boldsymbol{e}^\top,$$

as $H_{GRS}\boldsymbol{c}^\top = \boldsymbol{0}$. We can therefore write the syndrome entries using the error word as

$$S_i = \sum_{j=0}^{n-1} e_j v_j \alpha_j^i, \qquad i = 0, \dots, d-2,$$

or equivalently as

$$S_i = \sum_{j \in T} e_j v_j \alpha_j^i, \qquad i = 0, \ldots, d-2.$$

The syndrome polynomial can be then expressed as

$$S = \sum_{i=0}^{d-2} x^i \sum_{j \in T} e_j v_j \alpha_j^i = \sum_{j \in T} e_j v_j \sum_{i=0}^{d-2} (\alpha_j x)^i.$$

Consider now the ring $\mathbb{F}_q[x]/x^{d-1}$, which is the ring of polynomials over $\mathbb{F}$ of degree less than $d-1$. It is also the ring of remainders after dividing by x^{d-1}, therefore we can omit any monomials having x to the power of $d-1$ or higher The multiplicative inverse of the polynomial $\sum_{i=0}^{d-2}(\alpha_j x)^i$ in this ring is the polynomial $1 - \alpha_j x$, as

$$(1 - \alpha_j x) \sum_{i=0}^{d-2} (\alpha_j x)^i = 1 - (\alpha_j x)^{d-1} \equiv 1 \qquad (\mathrm{mod}\ x^{d-1}).$$

We can therefore express a relation for the syndrome polynomial

$$S \equiv \sum_{j \in T} e_j v_j (1 - \alpha_j x)^{-1} \qquad (\mathrm{mod}\ x^{d-1}).$$

3.2 Key Equation

We will now form relations, which are together denoted as the key equation of GRS decoding. We start by defining two polynomials. The *error locator polynomial* Λ (ELP) is defined as

$$\Lambda = \prod_{i \in T} (1 - \alpha_i x),$$

and the *error evaluator polynomial* Γ (EEP) is defined as

$$\Gamma = \sum_{i \in T} e_i v_i \prod_{j \in T \setminus \{i\}} (1 - \alpha_j x)$$

(a product over an empty set is considered to be 1).
The degrees of the polynomials are $\deg \Lambda = |T| = t \leq \frac{1}{2}(d-1)$ and $\deg \Gamma < t$. This gives us the first relation

$$\deg \Gamma < \deg \Lambda \leq \frac{1}{2}(d-1).$$

Next we observe that $\Lambda(\alpha_i^{-1}) = 0 \iff i \in T$. It is legal to substitute the multiplicative inverse of α_i as all α_i are nonzero. Therefore the roots of ELP tell us where the error positions are. The EEP evaluated at α_i^{-1} for $i \in T$ gives us

$$\Gamma(\alpha_i^{-1}) = e_i v_i \prod_{j \in T \setminus \{i\}} (1 - \alpha_j \alpha_i^{-1}).$$

We see that Γ does not share any of t roots of Λ, therefore we obtain our second relation, which is that $GCD(\Lambda, \Gamma) = 1$. In the special case of no errors occurring,

we get $\Gamma = 0, \Lambda = 1$ and $S = 0$.
The third relation is given by the mutual relation of ELP and EEP by

$$\begin{aligned}\Gamma =& \sum_{i \in T} e_i v_i \prod_{j \in T \setminus \{i\}} (1 - \alpha_j x) \\ \equiv& \sum_{i \in T} e_i v_i (1 - \alpha_i x)^{-1} \prod_{j \in T} (1 - \alpha_j x) \\ \equiv& \Lambda \sum_{i \in T} e_i v_i (1 - \alpha_j x)^{-1} \qquad (\mathrm{mod}\, x^{d-1}).\end{aligned}$$

Using the congruence with the syndrome polynomial we get the last relation

$$\Lambda S \equiv \Gamma \qquad (\mathrm{mod}\, x^{d-1}).$$

To summarize, the *key equation* of GRS decoding is given by three relations:

$$\begin{aligned}& GCD(\Lambda, \Gamma) = 1, \\ & \deg \Gamma < \deg \Lambda \leq \frac{1}{2}(d-1), \\ & \Lambda S \equiv \Gamma \qquad (\mathrm{mod}\, x^{d-1}).\end{aligned}$$

The next step for in the decoding algorithm is solving the key equation for Λ. Once we know it, we can check which elements $\alpha_0^{-1}, \ldots, \alpha_{n-1}^{-1}$ are roots of Λ, which will give us the set of errors T. After that, the equations

$$S_i = \sum_{j \in T} e_j v_j \alpha_j^i, \qquad i = 0, \ldots, d-2$$

become linear in e_j. We can then solve this system of equations to obtain the error values. There is more effective method of finding the error values than through Gaussian elimination, which will be described later in this chapter. But now, we continue with a method for solving the key equation.

3.3 Solving the Key Equation

In this section we will show that we can compute the polynomials Λ and Γ by applying the Partial Euclidean Algorithm to certain polynomials.

Lemma 3.1. *Using the notation of EEA, suppose that v and r are two nonzero polynomials over $\mathbb{F}_q$ satisfying the following conditions:*

1. $GCD(v, r) = 1$,
2. $\deg v + \deg r < \deg r_0$,
3. $vr_1 \equiv r (mod\ r_0)$.

Then there is an index $h \in \{0, 1, \ldots, m+1\}$ and a constant $c \in \mathbb{F}_q$ such that

$$v = c \cdot v_h \qquad and \qquad r = c \cdot r_h.$$

Proof. The degrees of r_i decrease strictly and from the second condition we have $\deg r < \deg r_0$. Therefore there exists a unique value $h \geq 1$ of the index i for which $\deg r_h \leq \deg r < \deg r_{h-1}$. Step h of the EEA, as all the other steps, satisfies

$$u_h r_0 + v_h r_1 = r_h.$$

By the third condition there exists a polynomial u such that

$$ur_0 + vr_1 = r.$$

Multiplying the latter two equations by v or v_h respectively and subtracting the resulting equations, we obtain

$$(vu_h - v_h u)r_0 = vr_h - v_h r.$$

Now from the second condition and the degree constrains from the beginning of the proof, we get

$$\deg v + \deg r_h \leq \deg v + \deg r < \deg r_0.$$

We already saw that $\deg r < \deg r_{h-1}$, and from 1.1(1) for $i = h - 1$ we get $\deg v_h + \deg r_{h-1} = \deg r_0$. Using these relations, we obtain

$$\deg v_h + \deg r = \deg r_0 - \deg r_{h-1} + \deg r < \deg r_0 - \deg r + \deg r = \deg r_0.$$

This means, that the right-hand side of the expression

$$(vu_h - v_h u)r_0 = vr_h - v_h r$$

has degree less than $\deg r_0$. But as the left-hand side is a multiple of r_0, the right-hand side must be equal to zero, meaning $vr_h = v_h r$. Lemma 1.1(1) implies that $\deg v_h \geq 0$ (as degrees of r_i decrease strictly), therefore both sides of $vr_h = v_h r$ are nonzero. From this equation and the first condition of this lemma we see that r divides r_h. And as $\deg r_h \leq \deg r$, there is a constant c such that $r = c \cdot r_h$. This also implies that $v = c \cdot v_h$, which proves the lemma. □

Based on Lemma 3.1 we can solve the key equation using the EEA. The relations that form the key equation imply the conditions of the lemma. We can therefore apply the EEA to polynomials $r_0 = x^{d-1}$ and $r_1 = S$ to produce $\Lambda = c \cdot v_h$ and $\Gamma = c \cdot r_h$, where the constant c is set so that $\Lambda(0) = 1$. We still need to determine the value of h though, as in Lemma 3.1 we are setting it as if we already knew the output polynomials. Next lemma will tell us how to set the index h for unknown polynomials.

Lemma 3.2. *Let v and r be as in Lemma 3.1. Furthermore assume that*

$$\deg v \leq \frac{1}{2}\deg r_0 \qquad \textit{and} \qquad \deg r < \frac{1}{2}\deg r_0.$$

Then the value h in Lemma 3.1 is the unique index for which the remainders of the EEA satisfy

$$\deg r_h < \frac{1}{2}\deg r_0 \leq \deg r_{h-1}.$$

Proof. A smaller index i would result in a polynomial $c \cdot r_i$ whose degree is too large. On the other hand, by Lemma 1.1(1) we have for every $i > h$

$$\deg v_i \geq \deg v_{h+1} = \deg r_0 - \deg r_h > \frac{1}{2} \deg r_0.$$

So for every $i > h$ we would get a polynomial $c \cdot v_i$ whose degree is too large. □

The restrictions in Lemma 3.2 are satisfied by the degree restrictions of the key equation. These restrictions also give us the stopping threshold for the index h. We can therefore solve the key equation by applying the PEA to polynomials $r_0 = x^{d-1}$ and $r_1 = S$ and the stopping threshold $\frac{1}{2}(d-1)$.

3.4 Uniqueness of the Solution

In this section, we show that the polynomials Λ and Γ obtained by the PEA are unique, given the relations from the key equation.

Lemma 3.3. *Polynomials Λ and Γ returned by the PEA as described above are up to scaling by a constant $c \in \mathbb{F}_q^*$ the unique solution to the key equation.*

Proof. We will show that if polynomials $\lambda, \gamma \in \mathbb{F}_q[x]$ satisfy the key equation, then

$$\lambda = \Lambda \cdot c \qquad \text{and} \qquad \gamma = \Gamma \cdot c.$$

The key equation is given by three relations

$$\begin{aligned} &GCD(\Lambda, \Gamma) = 1, \\ &\deg \Gamma < \deg \Lambda \leq \frac{1}{2}(d-1), \\ &\Lambda S \equiv \Gamma \pmod{x^{d-1}}. \end{aligned}$$

As $\Lambda(0) = 1$, Λ has a multiplicative inverse in the ring $\mathbb{F}_q[x]/x^{d-1}$. From the third relation we obtain

$$S \equiv \Gamma \Lambda^{-1} \pmod{x^{d-1}}.$$

Therefore λ and γ can satisfy the key equation only if it holds that

$$\lambda \Gamma \Lambda^{-1} \equiv \gamma \pmod{x^{d-1}},$$

or equivalently

$$\lambda \Gamma \equiv \Lambda \gamma \pmod{x^{d-1}}.$$

As we suppose the degrees constraints for all polynomials in the latter congruence, the degrees of both side of the congruence are lower than $d-1$. We can therefore rewrite it as an equality $\lambda\Gamma = \Lambda\gamma$, from which we get that $\Lambda \mid \lambda$, therefore $\lambda = c \cdot \Lambda$ for some nonzero polynomial $c \in \mathbb{F}[x]$. We know from Lemma 3.2 that the index of the EEA algorithm is unique, and therefore the degrees of the received polynomials are also, therefore $c \in \mathbb{F}_q^*$, which gives us the desired result

$$\lambda = \Lambda \cdot c \qquad \text{and} \qquad \gamma = \Gamma \cdot c.$$

□

3.5 Computing of the Error Values

After we found Λ and Γ and the error locations by checking whether α_i^{-1} is a root of Λ, we can now compute the error values using this efficient way, which we will now describe. We recall the *formal derivative* of the polynomial $a = \sum_{i=0}^{m} a_m x^m$ being

$$a' = \sum_{i=1}^{m} i a_i x^{i-1}.$$

The formal derivative of the product of two polynomials a and b obeys the rule (as in Lang [2005])

$$(ab)' = a'b + ab'.$$

By repeated application of this rule to the polynomial Λ, we get its formal derivative

$$\Lambda' = \sum_{i \in T} (-\alpha_i) \prod_{j \in T \setminus \{i\}} (1 - \alpha_j x).$$

So for every $\ell \in T$ we have

$$\begin{aligned}
\Lambda'(\alpha_\ell^{-1}) =& \sum_{i \in T} (-\alpha_i) \prod_{j \in T \setminus \{i\}} (1 - \alpha_j \alpha_\ell^{-1}) \\
=& -\alpha_\ell \prod_{j \in T \setminus \{\ell\}} (1 - \alpha_j \alpha_\ell^{-1}) + \sum_{i \in T \setminus \{\ell\}} (-\alpha_i)(1 - \alpha_\ell \alpha_\ell^{-1}) \prod_{j \in T \setminus \{i,\ell\}} (1 - \alpha_j \alpha_\ell^{-1}) \\
=& -\alpha_\ell \prod_{j \in T \setminus \{\ell\}} (1 - \alpha_j \alpha_\ell^{-1}).
\end{aligned}$$

Furthermore, for every $\ell \in T$ we get

$$\Gamma(\alpha_\ell^{-1}) = \sum_{i \in T} e_i v_i \prod_{j \in T \setminus \{i\}} (1 - \alpha_j \alpha_\ell^{-1}) = e_\ell v_\ell \prod_{j \in T \setminus \{\ell\}} (1 - \alpha_j \alpha_\ell^{-1}),$$

using the same steps as for substituting to Λ'. This gives us a way to express the error values.

Lemma 3.4. *The error values e_ℓ are computed as*

$$e_\ell = \begin{cases} -\frac{\alpha_\ell}{v_\ell} \cdot \frac{\Gamma(\alpha_\ell^{-1})}{\Lambda'(\alpha_\ell^{-1})}, & \text{if } \Lambda(\alpha_\ell^{-1}) = 0 \\ 0 & \text{otherwise} \end{cases}, \qquad \ell = 0, \ldots, n-1.$$

This formula is known as the ***Forney's algorithm****.*

The correctness of the formula is given by the argumentation above.
Note that we gave the formula only for Λ and Γ, which is sufficient, as by Lemma 3.3 the polynomials returned by the PEA are unique up to scaling by a constant $c \in \mathbb{F}_q^*$. And as the formal derivation of polynomial a satisfies the rule $(c \cdot a)' = c \cdot a'$ (Lang [2005]), so the constants in the formula cancel each other out. So even though the original Λ is defined so that $\Lambda(0) = 1$, we can work with its multiple by any constant $c \in \mathbb{F}_q^*$.

3.6 Overview of the Algorithm

The algorithm can be summarized as follows.

Algorithm 4: Current Algorithm

Input : received word $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{F}_q^n$
Output: error word $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1}) \in \mathbb{F}_q^n$

1 **Syndrome computation:** Compute the polynomial $S = \sum_{i=0}^{d-2} S_i x^i$, where

$$S_i = \sum_{j=0}^{n-1} b_j v_j \alpha_j^i, \qquad i = 0, \ldots, d-2.$$

2 **Solving the key equation:** Apply the Partial Euclidean Algorithm to polynomials x^{d-1}, S and stopping threshold $\frac{1}{2}(d-1)$, receiving

$$ux^{d-1} + \Lambda S = \Gamma.$$

3 **Forney's algorithm:** Compute the error locations and their values by

$$e_i = \begin{cases} -\frac{\alpha_i}{v_i} \cdot \frac{\Gamma(\alpha_i^{-1})}{\Lambda'(\alpha_i^{-1})}, & \text{if } \Lambda(\alpha_i^{-1}) = 0 \\ 0 & \text{otherwise} \end{cases}, \qquad i = 0, \ldots, n-1.$$

output $\mathbf{e}$

3.7 Time Complexity of Algorithm 4

The complexity of these steps can be found in Stanovský and Barto [2011]. The algorithm consists of three steps, all of which are of complexity $O(n^2)$. During the syndrome computation, we only multiply $3n$ elements $d-1$ times, so the complexity is of the $O(n^2)$ class. The computational time of the PEA is also $O(n^2)$, as can be found in any scripts of computer algebra. In the Forney's algorithm, we again perform steps that are linear in n (both substituting to a polynomial and multiplying in the formula), and the number of steps is n, which gives us again the complexity of $O(n^2)$. Therefore, the overall complexity of Algorithm 4 is $O(n^2)$.

3.8 Example

Let us consider the field $\mathbb{F}_8$ as described in Example 2.3. This time we consider a code defined by the kernel of the matrix H, where

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha+1 & \alpha^2+\alpha & \alpha^2+\alpha+1 & \alpha^2+1 \\ 1 & \alpha^2 & \alpha^2+\alpha & \alpha^2+1 & \alpha & \alpha+1 & \alpha^2+\alpha+1 \end{pmatrix}.$$

This is as code with parameters $n = 7, k = 4$ and the distance $d = n - k + 1 = 4$. Therefore this code is able to correct up to 1 error.

Consider now the codeword $\boldsymbol{c} = (\alpha+1, \alpha^2+1, \alpha^2+\alpha, 1, 1, 1, 1)$. Suppose that the error $\boldsymbol{e} = (0, 0, 0, \alpha, 0, 0, 0)$ occurred during the transmission, resulting in the received word being $\boldsymbol{b} = \boldsymbol{c} + \boldsymbol{e} = (\alpha+1, \alpha^2+1, \alpha^2+\alpha, \alpha+1, 1, 1, 1)$. To decode this received word, we proceed as described above.
First, we compute the syndrome polynomial $S = \sum_{i=0}^{2} S_i x^i$, where S_i are given as

$$\begin{pmatrix} S_0 \\ S_1 \\ S_2 \end{pmatrix} = H\boldsymbol{b}^\top.$$

The resulting syndrome polynomial is $S = x^2 + (\alpha^2+\alpha)x + \alpha$.
Next step is to solve the key equation by applying the PEA to polynomials $x^{(d-1)} = x^3$ and S and the stopping threshold $\frac{1}{2}(d-1) = 3/2$, receiving the relation

$$ux^3 + \Lambda S = \Gamma.$$

We obtain the polynomials $\Lambda = x + (\alpha^2+\alpha)$ and $\Gamma = (\alpha^2+\alpha+1)$. By extensive search of which $(\alpha^i)^{-1}, i = 0, \ldots, 6$ is a root of Λ, we get that the only root is at α^{-3}, which corresponds to error position e_3 to be nonzero.
Last step is to compute the value of the error in this position by applying the relation

$$e_3 = \alpha^3 \frac{\Gamma(\alpha^{-3})}{\Lambda'(\alpha^{-3})} = \alpha.$$

The algorithm found the error values $(0, 0, 0, \alpha, 0, 0, 0)$, which is exactly the error $\boldsymbol{e}$. Therefore, the decoded word corresponds to the original codeword $\boldsymbol{c}$, so the decoding was successful.

4. GAO Algorithm

In this chapter we will describe another algorithm described by Shuhong Gao in Gao [2003]. We will use the notation as described in the first chapter, working with the finite field $\mathbb{F}_q$. We will encode messages of length k to codewords of length n, for which we fix any n different field elements $a_i \in \mathbb{F}_q, i = 0, \ldots, n-1$, which will be used during the evaluation of the message polynomial.

4.1 Decoding

We consider received word $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{F}_q^n$, which we get by transmission through a channel from the codeword $\mathbf{c}$. We suppose that t errors occurred during the transmission, where $t \leq (d-1)/2$ ($d = n - k + 1$ being the distance of the code). We again denote T the set of error locations, meaning $T = \{i \mid b_i \neq c_i, i = 0, \ldots, n-1\}$, therefore $|T| = t \leq (d-1)/2$. We precompute the polynomial

$$g_0 = \prod_{i=0}^{n-1} (x - a_i) \in \mathbb{F}_q[x].$$

Note that g_0 is known and does not have to be computed for many cases. If, for instance, $n \mid (q-1)$ and $a_0, \ldots, a_{n-1}$ form a multiplicative group in $\mathbb{F}_q$, then $g_0 = x^n - 1$, or if $q = n$, then $g_0 = x^q - x$.
To decode b we proceed as described in the following algorithm.

Algorithm 5: GAO Algorithm

Input : received vector $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{F}_q^n$
Output: message polynomial $f = m_0 + m_1 x + \ldots + m_{k-1} x^k$ or "Decoding failure"

1 **Interpolation:** Find the unique polynomial $g_1 \in \mathbb{F}_q[x]$ of degree $\leq n-1$ which satisfies

$$g_1(a_i) = b_i, \;\; i = 0, 1, \ldots, n-1.$$

2 **Partial GCD:** Apply the Partial Euclidean Algorithm to polynomials g_0, g_1 and stopping threshold $\frac{1}{2}(n+k)$, receiving

$$u g_0 + v g_1 = g.$$

3 **Long division:** Divide g by v getting

$$g = f_1 v + r,$$

where $\deg r < \deg v$.
4 **if** $r = 0$ *and deg* $f_1 < k$ **then**
5 | output f_1
6 **else**
7 | output "Decoding failure"
8 **end**

"Decoding failure" means, that the algorithm was not able to decode the received word as there is no codeword with distance $< d/2$ from the received word. Also the algorithm might output wrong message polynomial in the case of so many errors happening that the received word is closer to another codeword than the original. So we can take the output word of the algorithm as the correct one only if we suppose that no more than $(d-1)/2$ errors occured during the transmission.

4.2 Correctness of Algorithm 5

In this section we will show why Algorithm 5 works correctly. We will be using the notation for the EEA as described in the beginning of the work.

Lemma 4.1. *Suppose two nonzero polynomials $r_0, r_1 \in \mathbb{F}_q[x]$ and the notation of the EEA. Then*

$$u_{m+1} = (-1)^{m+1}\frac{r_1}{r_m},\ \ v_{m+1} = (-1)^m \frac{r_0}{r_m}.$$

Proof. The relation for calculating u_i, v_i written in the matrix form is

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{bmatrix}.$$

By iterating this matrix equation i times we get

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -q_{i-1} \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 \\ 1 & -q_1 \end{bmatrix} \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix}.$$

From the initial choice of $(u_0, u_1) = (1, 0), (v_0, v_1) = (0, 1)$ we get

$$\begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Using this we see the determinants are

$$u_i v_{i+1} - u_{i+1} v_i = \det \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} = \prod_{j=1}^{i} \det \begin{bmatrix} 0 & 1 \\ 1 & -q_j \end{bmatrix} = (-1)^i.$$

We can now use this determinant and formula from linear algebra for computing inverse matrix $A^{-1} = \frac{1}{\det A}\mathrm{adj}A$, where $\mathrm{adj}A$ is the adjugate matrix of A, to get the inverse matrix

$$\begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix}^{-1} = (-1)^i \begin{bmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{bmatrix}.$$

We can write the relations for r_i and r_{i+1} in the matrix form as

$$\begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix}, i = 0, \ldots, m.$$

We can then express r_0, r_1, using the earlier computed inverse matrix, to get

$$\begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{bmatrix}^{-1} \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix} = (-1)^i \begin{bmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{bmatrix} \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}, i = 0, \ldots, m.$$

By choosing $i = m$ we get the expressions

$$\begin{aligned} r_0 =& (-1)^m v_{m+1} r_m \\ r_1 =& (-1)^{m+1} u_{m+1} r_m \end{aligned}$$

as $r_{m+1} = 0$. By adjusting these expressions by isolating u_{m+1} or v_{m+1} respectively, we get the relations formulated in the Lemma. □

Lemma 4.2. *Let $g_0 = w_0 r_0 + \epsilon_0$ and $g_1 = w_0 r_1 + \epsilon_1$, where $GCD(r_0, r_1) = 1$ and*

$$\deg\ r_i \leq t,\ \deg\ \epsilon_i \leq l,\ \ i = 0, 1.$$

Suppose that d_0 satisfies $\deg w_0 \geq d_0 > l + t$. After applying PEA to g_0 and g_1 and stopping threshold d_0, we get the expression

$$u g_0 + v g_1 = g.$$

Then $\exists \alpha \in \mathbb{F}_q^*$ that satisfies

$$u = -\alpha r_1,\ \ v = \alpha r_0.$$

Proof. We will show that EEA computes the same sequence of quotients for both pairs r_0, r_1 and g_0, g_1. We suppose that

$$r_{i-1} = q_i r_i + r_{i+1}, \qquad \deg r_{i+1} < \deg r_i, \qquad i = 1, \ldots, m,$$

where $r_{m+1} = 0$ and $r_m \in \mathbb{F}_q^*$ as $GCD(r_0, r_1) = 1$.
Using the notation for the EEA from the beginning of the work and by Lemma 1.1(3) and (4) we have

$$\deg u_i \leq \deg r_1 \leq t, \qquad \deg v_i \leq \deg r_0 \leq t$$

and from Lemma 4.1 we get

$$u_{m+1} = (-1)^{m+1} \frac{r_1}{r_m},\ \ v_{m+1} = (-1)^m \frac{r_0}{r_m}.$$

We now define

$$g_i = u_i g_0 + v_i g_1, \qquad 2 \leq i \leq m + 1.$$

It holds

$$g_{i-1} = q_i g_i + g_{i+1}, \qquad 1 \leq i \leq m$$

because by substituting the definition of g_i and g_{i+1} we get

$$\begin{aligned} g_{i-1} =& q_i g_i + g_{i+1} = q_i(u_i g_0 + v_i g_1) + u_{i+1} g_0 + v_{i+1} g_1 \\ =& (u_i q + u_{i+1}) g_0 + (v_i q + v_{i+1}) g_1 \\ =& u_{i-1} g_0 + v_{i-1} g_1, \end{aligned}$$

which is the definition of g_{i-1}.
We want to show that the degrees of $g_1, \ldots, g_{m+1}$ decrease strictly. It holds for $i = 0, \ldots, m + 1$ that

$$\begin{aligned} g_i =& u_i(w_0 r_0 + \epsilon_0) + v_i(w_0 r_1 + \epsilon_1) \\ =& w_0(u_i r_0 + v_i r_1) + (u_i \epsilon_0 + v_i \epsilon_1) \\ =& w_0 r_i + (u_i \epsilon_0 + v_i \epsilon_1). \end{aligned}$$

By our assumptions we have for $i = 0, \ldots, m+1$

$$\deg \epsilon_0 \leq l, \deg u_i \leq \deg r_1 \leq t, \qquad \deg \epsilon_1 \leq l, \deg u_i \leq \deg r_0 \leq t.$$

Therefore it holds that

$$\deg(u_i\epsilon_0 + v_i\epsilon_1) \leq l + t < d_0 \leq \deg w_0 \qquad i = 0, \ldots, m+1,$$

and we have

$$\deg g_i = \deg w_0 + \deg r_i \geq \deg w_0 \geq d_0 > l + t, \qquad i = 0, \ldots, m,$$

and

$$\deg g_{m+1} = \deg(u_{m+1}\epsilon_0 + v_{m+1}\epsilon_1) \leq l + t.$$

Because we assumed that degrees of $r_1, \ldots, r_m$ decrease strictly, also the degrees of $g_1, \ldots, g_m$ decrease strictly. This means that the quotient sequence $q_1, \ldots, q_m$ is the same for both pairs r_0, r_1 and q_0, g_1, up to step m. This then implies that the sequence of u_i and v_i is also the same. Step m is also the first time that the remainder satisfies $\deg g_{m+1} < d_0$, and at this step we receive

$$u_{m+1}g_0 + v_{m+1}g_1 = g_{m+1} = -\alpha r_1 g_0 + \alpha r_0 g_1,$$

where $\alpha = (-1)^{m+1}/r_m$, which is the desired result. □

Theorem 4.3. *If the received vector* $\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})$ *has distance at most* $(d-1)/2$ *from the codeword* $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ *defined by the message polynomial* f*, then the Algorithm 5 returns* f*.*

Proof. Suppose that the received vector $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ has the distance $t \leq (d-1)/2$ from the unique codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ defined by f. We define the *error locator polynomial* as

$$w(x) = \prod_{i \in T}(x - a_i),$$

T denoting the set of error locations, so $\deg w = t$. We name w_0 the cofactor of w in g_0, so $g_0 = w_0 w$.
We define the unique polynomial $\tilde{w} \in \mathbb{F}_q[x]$ with degree $< t$ that satisfies

$$\tilde{w}(a_i) = (b_i - c_i)/w_0(a_i), \;\; for i \in T.$$

Then $GCD(w, \tilde{w}) = 1$, and $g_1 = w_0 \cdot \tilde{w} + f$, as both sides have degree less than n and have same value b_i when evaluated in a_i for $i = 0, 1, \ldots, n-1$, because

$$w_0(a_i)\tilde{w}(a_i) + f(a_i) = 0 + f(a_i) = b_i = g_1(a_i).$$

Let $d_0 = (n+k)/2$. Note that

$$\deg w_0 = \deg g_0 - \deg w = n - t \geq n - \frac{n-k}{2} = d_0 > k - 1 + t \geq \deg f + \deg w.$$

Then by Lemma 4.2 we have $u = -\alpha\tilde{w}$ and $v = \alpha w$ for some $\alpha \in \mathbb{F}_q^*$. We can write g as

$$\begin{aligned} g =& ug_0 + vg_1 = -\alpha\tilde{w}g_0 + \alpha w g_1 \\ =& -\alpha\tilde{w}g_0 + \alpha w(w_0\tilde{w} + f) = \alpha w f = vf. \end{aligned}$$

This means that in the long division step of Algorithm 3 the remainder is zero and the quotient $f_1 = f$ as both have degree $< k$.
On the other hand, suppose that in step 3 the algorithm returns a polynomial f_1, which defines a codeword as it has degree $< k$. The identity in the partial GCD step implies

$$ug_0 = g - vg_1 = vf_1 - vg_1 = v(f_1 - g_1),$$

which means that

$$v(a_i)(f_1(a_i) - g_1(a_i)) = 0, \;\; i = 0, 1, \ldots, n-1.$$

But $\deg v = t \leq (d-1)/2$, so $f_1(a_i) = g_1(a_i)$ for at least $n - t \geq n - (d-1)/2$ values of i. Therefore $\mathbf{b}$ has distance $\leq (d-1)/2$ from the codeword defined by f_1, which means that the codeword defined by f_1 is the unique codeword with distance $\leq (d-1)/2$ from the received word. □

4.3 Time Complexity of Algorithm 5

The algorithm consists of three steps: Interpolation, Partial GCD and Long division. The complexity of these algorithms can be found in Stanovský and Barto [2011].
The complexity of interpolation of a polynomial of deg $\leq n-1$ using Garner or Lagrange algorithm is $O(n^2)$.
The second step is the Partial Euclidean Algorithm used on 2 polynomials of deg $\leq n$, which has the complexity of $O(n^2)$.
The last step is division of the polynomial $g(x)$. $\deg g(x) \leq \frac{1}{2}(n+k) \leq n$, as the codeword cannot be shorter than the message to be able to correct any errors. Therefore using algorithm for fast division using formal power series and fast Fourier transformation, we get the complexity of $O(n \log n)$.
Total time complexity of the algorithm is

$$O(n^2) + O(n^2) + O(n \log n) = O(n^2).$$

4.4 Example

Let us assume the same situation as in Example 2.3, which means the message $\mathbf{m} = (\alpha, \alpha^2, \alpha^2 + \alpha + 1)$, to which we associate corresponding message polynomial $f = \alpha + \alpha^2 x + (\alpha^2 + \alpha + 1)x^2$, the codeword $\mathbf{c} = (\alpha, 0, 0, \alpha + 1, \alpha, 1, \alpha + 1, 1)$ and the transmission error $\mathbf{e} = (\alpha, 1, 0, 0, 0, 0, 0, 0)$, which results in the received word $\mathbf{b} = (0, 1, 0, \alpha + 1, \alpha, 1, \alpha + 1, 1)$.
To decode the message, this time using the GAO algorithm, we first consider the polynomial

$$g_0 = \prod_{a \in \mathbb{F}_8} (x - a) = x^8 + x \in \mathbb{F}_8[x].$$

Note that in this case we actually do not need to compute the polynomial as we are using all elements of the field $\mathbb{F}_8$ to compute it and $8 = 2^3$ is a prime power, therefore the polynomial has the form of $x^{2^3} - x = x^8 + x$.

We now perform the steps of the algorithm:
1. Interpolation: We compute the polynomial

$$g_1 = x + (\alpha^2 + \alpha)x^3 + (\alpha + 1)x^4 + \alpha^2 x^5 + \alpha x^6 + (\alpha + 1)x^7.$$

2. Partial GCD: Perform the PEA on polynomials g_0, g_1 and stopping threshold $1/2(8 + 3) = 11/2$, receiving

$$\begin{aligned} g =& (\alpha^2 + \alpha + 1)x + (\alpha + 1)x^2 + (\alpha + 1)x^3 + \alpha x^4 \\ u =& (\alpha^2 + \alpha + 1) + (\alpha^2 + 1)x \\ v =& (\alpha^2 + \alpha)x + (\alpha + 1)x^2 \end{aligned}$$

satisfying

$$ug_0 + vg_1 = g.$$

3. Long division: Divide g by v getting

$$f_1 = \alpha + \alpha^2 x + (\alpha^2 + \alpha + 1)x^2, \quad r = 0$$

satisfying

$$g = f_1 v + r.$$

As the remainder $r = 0$ and deg $f_1 < 3$, the algorithm returns the polynomial f_1, which corresponds to the message polynomial f of the sent message, so the decoding was successful.

Conclusion

At the end of this thesis, I would like to summarize the results of this work, as well as comment on my personal contribution. The thesis provides an in-depth description of three different decoding algorithms for Reed-Solomon codes, including the original view, as well as the modern approach and a another possible efficient way of decoding. I compiled the algorithms from various resources and united them under the same notation. I also described in detail every step of all algorithms, explaining sections of proofs, that were mostly skipped by original authors. Furthermore, I expanded on some ideas and propositions, that were not fully described, such are the Voting Algorithm or proof of the length of the period for binary representation of the field $\mathbb{F}_{2^N}$. I also made a comment about the time complexity of each algorithm, so their efficiency is easily comparable. And lastly, I made a working implementation of each of these algorithms and used them to provide basic examples, which show the process and the structure of decoding of each algorithm.

Bibliography

L. Barto and J. Tůma. *Lineární Algebra*. Scripts in development. 2023.

S. Gao. A new algorithm for decoding Reed-Solomon codes. *Communications, Information and Network Security*, 712:55–68, 2003.

S. Lang. *Undergraduate Algebra*. Springer, 2005. ISBN 9780387220253.

I.S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8, 1960.

R. M. Roth. *Introduction to Coding Theory*. Cambridge University Press, Cambridge, 2006. ISBN 978-0-521-84504-5.

D. Stanovský and L. Barto. *Počítačová Algebra*. MatfyzPress, 2011. ISBN 9788073783402.