ulm university universität
uulm

Decoding of Gabidulin Codes using Module Minimisation

Master's Thesis
of
David Mödinger

Institute of Communications Engineering
Ulm University

March 2015

M/2015/PU/1

ulm university universität
uulm

Institute of Communications Engineering

MASTER'S THESIS

Decoding of Gabidulin Codes using Module Minimisation

Explanation

Gabidulin codes are the rank metric analogues of Reed–Solomon codes and have a variety of applications like Random Linear Network Coding and cryptography. Interleaving of ℓ Gabidulin codes can be used to increase the decoding radius or to decrease the overhead in a lifting construction for Random Linear Network Coding.

Similar to many other algebraic codes, the main part of decoding Interleaved Gabidulin codes can be reduced to a shift register synthesis problem over a skew polynomial ring. Recent results have shown that this problem can be solved in $O(\ell\mu^2)$ time, where μ depends linearly on the code length, by extending the module minimisation approach known for decoding Reed–Solomon codes to skew polynomial rings.

In this thesis, these previous results are outlined and a variant of Alekhnovichs algorithm over skew polynomials is presented, which has a complexity of $O(\ell^\omega \mu^{1.69} \log \mu)$. This enables an ℓ-μ-tradeoff in the case of Interleaved Gabidulin codes and implies the existence of a sub-quadratic decoding algorithm of Gabidulin codes.

ulm university universität
uulm

Institute of Communications Engineering

Submission Date: 04.05.2015

Candidate: David Mödinger

Supervisors: B. Sc. Sven Puchinger, M. Sc. Sven Müelich

Examiner: Prof. Dr.-Ing. Martin Bossert

2nd Examiner: Prof. Dr.-Ing. Uwe Schöning

Catalog No.: M/2015/PU/1

I hereby confirm, that the presented Master's Thesis is an original work completed independently without inadmissible outside help.

Ulm, 04.05.2015

(David Mödinger)

Contents

1 Introduction

Gabidulin codes are the rank-metric equivalent of the widely used Reed–Solomon codes. Gabidulin codes are considered for random linear network coding, as in [KK08], and public key cryptography systems, as in [BL05]. Especially the application of random linear network coding inspires the use of interleaving in this thesis.

While there are different decoding schemes for Gabidulin codes, module minimisation brings a more straight forward approach, allowing easier proofs and understanding of the decoding process, and a generalised approach over a multitude of codes.

The goal of this thesis is to give an overview of the concept and add to it over generalised non–commutative polynomials and give a complexity estimate for the special case of Gabidulin codes. This is done by adding an algorithm to perform module minimisation over non–commutative polynomials and analysing its performance on decoding of (interleaved) Gabidulin codes.

Overview of the thesis

Chapter 2 introduces the necessary mathematical fundamentals used throughout the thesis. This includes an introduction to finite fields, linearised polynomials, Ore extensions and modules.

Chapter 3 gives a brief overview over the coding theory fundamentals of Reed–Solomon codes and, the main application of this thesis, Gabidulin codes.

In **Chapter 4** two known algorithms for non–commutative polynomials are examined: A review of the Mulders–Storjohann algorithm and the improved demand driven variant for decoding. This chapter also introduces the restrictions for the complexity analysis of the algorithms and common factors.

Based on the results of the previous chapter, in **Chapter 5** a divide and conquer variant of Mulders–Storjohann, Alekhnovich's algorithm, is converted to non–commutative polynomials. Alekhnovich's algorithm is then analysed under the re-

strictions for decoding resulting in a much more detailed complexity for this problem.

Chapter 6 examines the implication of a sub–quadratic decoding algorithm for non–interleaved Gabidulin codes, based on the analysis done in Chapter 4.

Chapter 7 summarizes the analysis of Chapter 4 and its implications in Chapter 6.

2 Mathematical Fundamentals

This chapter consists of two parts. In the first part, **Section 2.1** to **Section 2.4** necessary mathematical tools for this thesis are introduced, starting with finite fields in **Section 2.1**. Then **Section 2.2** introduces linearised polynomials, a special ring of polynomials, which are further generalised to Ore extensions in **Section 2.3**. In **Section 2.4** the concepts of modules and module minimisation are examined.

2.1 Finite Fields

Finite fields are a fairly well studied field of mathematics, so this section only revisits the notation for them. For a thorough introduction to finite fields refer to [LN96]. If not stated otherwise, the implicit operations are regular addition $\oplus$ and multiplication $\odot$.

2.1.1 Prime Fields

Let p be a prime, then a finite field with p elements exists and has the notation $\mathbb{F}_p$. It is isomorphic to the integers modulo p

$$\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z} = \{0, 1, \ldots, p-1\}.$$

2.1.2 Extension Fields

Let $q = p^n$ be a prime power for some $n \in \mathbb{N}$ and $f \in \mathbb{F}_q[x]$ be a polynomial which is irreducible over $\mathbb{F}_q$ with $\deg f = m \in \mathbb{N}$ then

$$\mathbb{F}_{q^m} \cong \mathbb{F}_q[x]/\langle f(x)\rangle$$

is a field with q^m elements and characteristic p. [LN96] explains that for any m such an irreducible polynomial exists and that all fields with equal cardinality are

isomorphic. So the structure of the field of size q^m is independent of the chosen polynomial.

Example 2.1.1 (Extension Field $\mathbb{F}_{2^3}$)
As an example we examine $\mathbb{F}_{2^3}$ defined by $m = 3$, $f(x) = x^3 + x + 1$ and $\mathbb{F}_2 = \{0, 1\}$. It is

$$\mathbb{F}_{2^3} = \mathbb{F}_2[x]/\langle x^3 + x + 1\rangle.$$

The elements of $\mathbb{F}_{2^3}$ can be determined by using a root of $f(x)$. As $f(x)$ is irreducible over $\mathbb{F}_2$ there are no roots within $\mathbb{F}_2$. We solve

$$\begin{aligned} & x^3 + x + 1 = 0 \\ \iff \quad & x^3 = -x - 1 \\ \iff \quad & x^3 = x + 1 \end{aligned}$$

and call a solution to this equation α. The resulting elements are listed in Table 2.1, they are transformed using the relation $\alpha^3 = \alpha + 1$.

Nr.	α Representation	Transformation	Representation
0	$= 0$		$= 0$
1	$= \alpha^0$		$= 1$
2	$= \alpha^1$		$= \alpha$
3	$= \alpha^2$		$= \alpha^2$
4	$= \alpha^3$	(defining relation)	$= \alpha + 1$
5	$= \alpha^4$	$= \alpha^3\alpha \ = (\alpha + 1)\alpha$	$= \alpha^2 + \alpha$
6	$= \alpha^5$	$= \alpha^3\alpha^2 = (\alpha + 1)\alpha^2$	$= \alpha^2 + \alpha + 1$
7	$= \alpha^6$	$= \alpha^3\alpha^3 = (\alpha + 1)^2$	$= \alpha^2 + 1$

Table 2.1: Elements of $\mathbb{F}_{2^3}$ using the relation $\alpha^3 = \alpha + 1$.

The ext Mapping

Using a basis β of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, $\mathbb{F}_{q^m}$ can be represented as a vector space over $\mathbb{F}_q$. A vector of elements can therefore be represented as a matrix over $\mathbb{F}_q$. This translation is done by the ext mapping.

Definition 2.1.1 (ext_β Mapping)
The ext *mapping is a function $\text{ext}_\beta : \mathbb{F}_{q^m}^n \longrightarrow \mathbb{F}_q^{m \times n}$ related to a basis β of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ with a fixed order. Let $\mathbf{a} \in \mathbb{F}_{q^m}^n$ and, $\beta \in \mathbb{F}_{q^m}^m$ with β a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. The mapping ext_β fulfils the equation*

$$\mathbf{a} = \beta \cdot \text{ext}_\beta(\mathbf{a}) = \beta \cdot \mathbf{A}. \tag{2.1}$$

The $\text{ext}_\beta(x)$ mapping might be noted as $\text{ext}(x)$ without explicitly mentioning the basis β, whenever the basis is implicit or not of importance to the problem at hand.

Equation (2.1) contains a trivial inverse for the ext mapping: Multiplication by the basis vector β. As ext conserves the structure of the underlying field, it is also an isomorphism. The mapping and its inverse allow switching between matrix- and field–representation for elements and vectors of elements. Depending on the problem, the more convenient representation might be used without explicit mention of the mapping.

Example 2.1.2 (ext **Mapping)**
Using some basis $\beta = (1, \alpha, \alpha^2)$ *and* $\gamma = (\alpha^3, \alpha^6, \alpha^5)$ *of* $\mathbb{F}_{2^3}$*, Table 2.2 shows the results for the* ext *mappings.*

$x \in \mathbb{F}_{2^3}$	Transformed Representation	$\text{ext}_\beta(x) \in \mathbb{F}_2^{3\times 1}$	$\text{ext}_\gamma(x) \in \mathbb{F}_2^{3\times 1}$
0	$= 0$	$\text{ext}_\beta \to (0,0,0)^T$	$\text{ext}_\gamma \to (0,0,0)^T$
1	$= 1$	$\text{ext}_\beta \to (1,0,0)^T$	$\text{ext}_\gamma \to (1,1,1)^T$
α^1	$= \alpha$	$\text{ext}_\beta \to (0,1,0)^T$	$\text{ext}_\gamma \to (0,1,1)^T$
α^2	$= \alpha^2$	$\text{ext}_\beta \to (0,0,1)^T$	$\text{ext}_\gamma \to (1,0,1)^T$
α^3	$= \alpha + 1$	$\text{ext}_\beta \to (1,1,0)^T$	$\text{ext}_\gamma \to (1,0,0)^T$
α^4	$= \alpha^2 + \alpha$	$\text{ext}_\beta \to (0,1,1)^T$	$\text{ext}_\gamma \to (1,1,0)^T$
α^5	$= \alpha^2 + \alpha + 1$	$\text{ext}_\beta \to (1,1,1)^T$	$\text{ext}_\gamma \to (0,0,1)^T$
α^6	$= \alpha^2 + 1$	$\text{ext}_\beta \to (1,0,1)^T$	$\text{ext}_\gamma \to (0,1,0)^T$

Table 2.2: The ext_β function mapping with $\beta = (1, \alpha, \alpha^2)$ and ext_γ function mapping with $\gamma = (\alpha^3, \alpha^6, \alpha^5)$. T symbolises transposition.

One important property of the ext mapping is the following behaviour of its inverse:

$$\text{ext}^{-1}(\mathbf{AB}) = \beta(\mathbf{AB}) = \beta\mathbf{A} \cdot \mathbf{B} = \text{ext}^{-1}(\mathbf{A}) \cdot \mathbf{B}.$$

2.2 Linearised Polynomials

While the general ring of polynomials over finite fields $\mathbb{F}[x]$ is a commutative ring, there are special polynomial rings which are not. One of those, is the ring of linearised polynomials, which will be used in Section 3.2 to describe Gabidulin codes.

Definition 2.2.1 (Linearised Polynomials)
The set of linearised polynomials (or q-polynomials) over a finite field $\mathbb{F}_{q^m}$ is the set

$$\mathcal{L}_q[x] = \{f(x)\}$$

with $a_i \in \mathbb{F}_{q^m}$, $d_f \in \mathbb{N}$ the degree of f and $f(x)$ of the form

$$\begin{aligned} f(x) &= \sum_{i=0}^{d_f} a_i x^{[i]} \\ &= a_0 x + a_1 x^q + a_2 x^{q^2} + \ldots + a_{d_f} x^{q^{d_f}}, \end{aligned}$$

where the operator $[i] = q^i$ is the q-power operator.

2.2.1 Operations on Linearised Polynomials

Although similar, linearised polynomials have some deviations from definitions of regular polynomials.

Degree

The degree of a linearised polynomial f is given by the function $\deg_q$ instead of deg and it is $\deg_q f = \log_q(\deg f)$, as the $\deg_q$ is defined by

$$f(x) = \sum_{i=0}^{d_f} f_i x^{[i]}, f_{d_f} \neq 0 \Rightarrow \deg_q f = d_f.$$

Addition

Just like regular polynomials, linearised polynomials are closed under addition and addition is carried out element–wise:

$$\sum_{i=0}^{d_f} f_i x^{[i]} + \sum_{i=0}^{d_g} g_i x^{[i]} = \sum_{i=0}^{\max\{d_f, d_g\}} (f_i + g_i) x^{[i]}.$$

Multiplication

Linearised polynomials differ from regular polynomials as they are not closed under regular multiplication. It is generally not true that

$$f, g \in \mathcal{L}_q[x] \Rightarrow f \cdot g \in \mathcal{L}_q[x].$$

Hence, symbolic multiplication in the form of composition is used for linearised polynomials:

$$f(x) \odot g(x) = f(g(x)).$$

2.2.2 Properties of Linearised Polynomials

There are some properties of linearised polynomials, that are important to note for later usage.

$\mathbb{F}_q$–Linearity [WZ13, Lemma 2.8]

For all $a_1, a_2 \in \mathbb{F}_q$ and all $b_1, b_2 \in \mathbb{F}_{q^m}$ and $f(x) \in \mathcal{L}_q[x]$ it holds, that

$$f(a_1 b_1 + a_2 b_2) = a_1 f(b_1) + a_2 f(b_2). \tag{2.2}$$

Roots of Linearised Polynomials [Ber84]

Roots of linearised polynomials are a linear subspace of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ and each root has a multiplicity of a power of q.

Minimal Subspace Polynomial [LN96]

Let $\mathcal{U}$ be a set of elements and $\langle \mathcal{U} \rangle$ the linear closure of $\mathcal{U}$, then the minimal subspace polynomial

$$M_{\mathcal{U}} = \prod_{\alpha \in \langle \mathcal{U} \rangle} (x - \alpha)$$

is a unique q-polynomial, containing every element of $\alpha \in \langle \mathcal{U} \rangle$ as a root with multiplicity one.

2.2.3 Modulo Minimal Subspace Polynomial

For the key equation for Gabidulin codes, it is useful to have a simple sufficient condition for modulo equivalence for minimal subspace polynomials.

Theorem 2.2.1
Two q-polynomials $f, g \in \mathcal{L}_q[x]$ are equivalent modulo a minimal subspace polynomial $M_\mathcal{U}$, if for all roots u of $M_\mathcal{U}$ they evaluate the same $f(u) = g(u)$.

$$(\forall u \in \mathbb{F}_{q^m} : M_\mathcal{U}(u) = 0 \Rightarrow f(u) = g(u)) \Rightarrow f \equiv g \mod M_\mathcal{U}.$$

Proof. As $\mathcal{L}_q[x]$ is a right–euclidean ring, as shown by [Ore33] (cf. Theorem 2.3.1), there always exist q-polynomials r and q for two q-polynomials a and c such that

$$\begin{aligned} a &= q \cdot c + r \\ \deg_q r &< \deg_q c. \end{aligned}$$

Assume $r \neq 0$. We consider the sets $M = \{x : M_\mathcal{U}(x) = 0\}$ and $R = \{x : r(x) = 0\}$, the sets of unique roots of $M_\mathcal{U}$ and r. Due to its construction, the minimal subspace polynomial $M_\mathcal{U}$ consists solely of unique factors with degree 1 and multiplicity 1. Therefore $M_\mathcal{U}$ has $\deg M_\mathcal{U}$ many roots and $M = q^{\deg_q M_\mathcal{U}}$. It is known that r has at most $q^{\deg_q r}$ many roots, so it follows, that $|R| <= q^{\deg_q(r)}$.

Suppose that $M \backslash R \neq \{\}$, so there exists a root of $M_\mathcal{U}$ that is not a root of r. For such a root x of $M_\mathcal{U}$:

$$\begin{aligned} 0 &= (a - b)(x) \\ &= (d(M_\mathcal{U}) + r)(x) \\ &= d(M_\mathcal{U}(x)) + r(x) \\ &\overset{(*)}{=} d(0) + r(x) \\ &= r(x) \neq 0. \end{aligned}$$

This is a contradiction. Step $(*)$ uses the condition of Theorem 2.2.1, which states, that a root of $M_\mathcal{U}$ is also a root of $a - b$. Thus $M \subseteq R \Leftrightarrow |M| \leq |R|$ has to be true. But this leads to

$$|M| \leq |R| \leq q^{\deg_q r} < q^{\deg_q M_\mathcal{U}} = |M|,$$

which is a contradiction, too. It follows, that $r = 0$. An equivalence modulo a linearised polynomial is equivalent to a division without remainder. The equivalence of f, g can therefore be rewritten:

$$\begin{aligned} & f \equiv g \mod M_\mathcal{U} \\ \Leftrightarrow \quad & \exists d \in \mathcal{L}_q[x] : f - g = d \cdot M_\mathcal{U}. \end{aligned}$$

As $r = 0$ the theorem is proven. □

2.3 Ore Extension

Ore extensions were introduced by [Ore33] and allow for a more general approach to non–commutative polynomials and not just linearised polynomials. They are a general form of non–commutative polynomials. A short introduction will help understand the usage of Ore extensions and why they are used.

Definition 2.3.1 (Ore Extension [Ore33])
Let $\mathbb{F}$ be a field, then $R = \mathbb{F}[x;\theta;\delta]$ is a non–commutative ring of polynomials. Let $f, g \in R,\ a \in \mathbb{F}$ and $n, m \in \mathbb{N}$, then

$$\begin{aligned} f \oplus g &= \sum_{i=0}^{max\{\deg f, \deg g\}} (f_i + g_i)x^i, \\ x \odot a &= \theta(a) \cdot x + \delta(a), \\ x^m x^n &= x^{m+n}, \end{aligned}$$

together with the field operations, are the operations of R.

The operation θ is called **conjugate** and the operation δ is called **derivative**. From Definition 2.3.1 [Ore33] derived some properties that need to be fulfilled by θ and δ. They are listed in Table 2.3 for $a, b \in \mathbb{F}$.

	$\oplus$	$\odot$
$\theta:$	$\theta(a+b) = \theta(a) + \theta(b)$	$\theta(ab) = \theta(a)\theta(b)$
$\delta:$	$\delta(a+b) = \delta(a) + \delta(b)$	$\delta(ab) = \theta(a)\delta(b) + \delta(a)b$

Table 2.3: Derived conditions for θ and δ by [Ore33].

For two monomials ax^m and bx^n with $a, b \in \mathbb{F}$ and $m, n \in \mathbb{N}$ this results in the product:

$$\begin{aligned} ax^m bx^n &= ax^{m-1} \underbrace{xb}_{Ore\odot rule} x^n \\ &= ax^{m-1}(\theta(b)x + \delta(b))x^n \\ &= a \underbrace{x^{m-1}\theta(b)}_{Further\ Ore\odot} x^{n+1} + a \underbrace{x^{m-1}\delta(b)}_{Further\ Ore\odot} x^n. \end{aligned}$$

The following example should help understand the usage of Ore multiplication for two small Ore polynomials.

Example 2.3.1 (Ore Extension Multiplication Behaviour)
Let $a = a_1x^2 + a_2x + a_3, b = b_1 \in \mathbb{F}[x;\theta;\delta]$ be two polynomials of some Ore extension. Then the multiplications of those polynomials, $c = a \cdot b$ and $d = b \cdot a$ have the form:

$$
\begin{aligned}
b \cdot a &= \underbrace{b_1a_1}_{=c_1} x^2 + \underbrace{b_1a_2}_{=c_2} x + \underbrace{b_1a_3}_{=c_3} \\
a \cdot b &= a_1x^2b_1 + a_2xb_1 + a_3b_1 \\
&= a_1(x \underbrace{xb_1}_{\odot \text{ by Ore rule}}) + a_2(\underbrace{xb_1}_{\odot \text{ by Ore rule}}) + a_3b_1 \\
&= a_1(x(\theta(b_1)x + \delta(b_1))) + a_2(\theta(b_1)x + \delta(b_1)) + a_3b_1 \\
&= a_1(\underbrace{x\theta(b_1)}_{\odot \text{ by Ore rule}} x + \underbrace{x\delta(b_1)}_{\odot \text{ by Ore rule}}) + a_2\theta(b_1)x + a_2\delta(b_1) + a_3b_1 \\
&= a_1((\theta(\theta(b_1))x + \delta(\theta(b_1)))x + \theta(\delta(b_1))x + \delta(\delta(b_1))) + a_2\theta(b_1)x + a_2\delta(b_1) + a_3b_1 \\
&= a_1(\theta^2(b_1)x^2 + \delta(\theta(b_1))x + \theta(\delta(b_1))x + \delta^2(b_1)) + a_2\theta(b_1)x + a_2\delta(b_1) + a_3b_1 \\
&= \underbrace{a_1\theta^2(b_1)}_{=d_1} x^2 + \underbrace{(a_1\delta(\theta(b_1)) + a_1\theta(\delta(b_1)) + a_2\theta(b_1))}_{=d_2} x + \underbrace{a_1\delta^2(b_1) + a_2\delta(b_1) + a_3b_1}_{=d_3}.
\end{aligned}
$$

This also shows that the multiplication is not commutative in general.

In [Ore33, Section 2] Ore shows that polynomials of Ore extensions are right divisible with remainder. Ore concludes, that Ore extensions support some right euclidean algorithm and therefore constitute an Euclidean domain. Left division is only possible if θ is an $\mathbb{F}-$automorphism, see [Ore33, Theorem 6].

2.3.1 Isomorphism of Linearised Polynomials and some Ore Extension

To be able to use the results acquired with Ore extensions, linearised polynomials need to be isomorphic to some Ore extension.

Theorem 2.3.1
The set of linearised polynomials over a finite field $\mathbb{F}$ is isomorphic to the Ore extension using the Frobenius automorphism $\theta(a) = a^q$ and the derivative $\delta(a) = 0$ for $a \in \mathbb{F}$:

$$\mathbb{F}[x; x^q; 0] \cong \mathcal{L}_q[x].$$

Proof. The conjugate θ and derivative δ fulfil the conditions of Table 2.3:

$$\begin{aligned}\theta(a+b) &= (a+b)^q \overset{(*)}{=} a^q + b^q = \theta(a) + \theta(b)\\ \delta(a+b) &= 0 = 0 + 0 = \delta(a) + \delta(b)\\ \theta(ab) &= (ab)^q = a^q b^q = \theta(a)\theta(b)\\ \delta(ab) &= \theta(a)\delta(b) + \delta(a)b = 0\end{aligned}$$

The only step needing extra caution is $(*)$, which is allowed due to the characteristic $p|q$ and the binomial theorem for finite fields.

Now consider $\phi : \mathbb{F}[x; x^q; 0] \longrightarrow \mathcal{L}_q[x]$, which is defined by the mapping

$$\phi : \sum_{i=0}^{d} a_i x^i \longrightarrow \sum_{i=0}^{d} a_i x^{[i]}.$$

For $\phi : \mathbb{F}[x; x^q; 0] \longrightarrow \mathcal{L}_q[x]$ with $u, v \in \mathbb{F}[x; x^q; 0]$ to be an isomorphism the following four conditions need to be fulfilled:

$$\begin{aligned}&\phi \text{ is bijective,} && (2.3)\\ &\phi(1_{\mathbb{F}[x;x^q;0]}) = 1_{\mathcal{L}_q[x]}, && (2.4)\\ &\phi(u+v) = \phi(u) + \phi(v), && (2.5)\\ &\phi(u \cdot v) = \phi(u) \cdot \phi(v). && (2.6)\end{aligned}$$

Equation (2.3), ϕ being bijective, is shown using the inverse ϕ^{-1}:

$$\begin{aligned}\phi^{-1} : \mathcal{L}_q[x] &\longrightarrow \mathbb{F}[x; x^q; 0]\\ \sum_{i=0}^{d} a_i x^{[i]} &\longrightarrow \sum_{i=0}^{d} a_i x^i.\end{aligned}$$

As $\phi^{-1}(\phi(a)) = a$ for $a \in \mathbb{F}[x; x^q; 0]$ and $\phi(\phi^{-1}(b)) = b$ with $b \in \mathcal{L}_q[x]$, it follows, that ϕ is bijective.

Equation (2.4) is fulfilled: $\phi(1_{\mathbb{F}[x;x^q;0]}) = \phi(x^0) = x^{[0]} = x = 1_{\mathcal{L}_q[x]}$.

Equation (2.5) can be shown by transformation:

$$\begin{aligned}\phi(u+v) &= \phi(\sum_{i=0}^{\max\{d_u,d_v\}}(u_i+v_i)x^i)\\ &= \sum_{i=0}^{\max\{d_u,d_v\}}(u_i+v_i)x^{[i]}\\ &= \sum_{i=0}^{d_u}u_ix^{[i]}+\sum_{i=0}^{d_v}v_ix^{[i]}\\ &= \phi(u)+\phi(v).\end{aligned}$$

Equation (2.6) can be shown by a slightly more complex transformation:

$$\begin{aligned}\phi(u\cdot v) &= \phi((\sum_{i=0}^{d_u}u_ix^i)\cdot(\sum_{j=0}^{d_v}v_jx^j))\\ &= \phi(\sum_{i=0}^{d_u}u_ix^i\sum_{j=0}^{d_v}v_jx^j) && = \phi(\sum_{i=0}^{d_u}u_i\sum_{j=0}^{d_v}x^iv_jx^j)\\ &= \phi(\sum_{i=0}^{d_u}u_i\sum_{j=0}^{d_v}v_j^qx^{i-1}v_jx^{j+1}) && = \phi(\sum_{i=0}^{d_u}u_i\sum_{j=0}^{d_v}v_j^{q^i}x^{i+j})\\ &= \sum_{i=0}^{d_u}u_i\sum_{j=0}^{d_v}v_j^{[i]}x^{[i+j]} && = \sum_{i=0}^{d_u}u_i\sum_{j=0}^{d_v}(v_jx^{[j]})^{[i]}\\ &\overset{(*)}{=} \sum_{i=0}^{d_u}u_i(\sum_{j=0}^{d_v}v_jx^{[j]})^{[i]}\\ &= \phi(u)(\phi(v))\\ &= \phi(u)\cdot\phi(v).\end{aligned}$$

Step $(*)$ follows due to the characteristic $p|q$ and the binomial theorem for finite fields, seen in the first step.

Thus ϕ is an isomorphism and Theorem 2.3.1 is true. □

Every result for Ore extensions can therefore be transformed and applied to linearised polynomials. As $\theta(a) = a^q$ is an automorphism, $\mathcal{L}_q[x]$is also left divisible.

2.4 Modules and Module Minimisation

Vectors of polynomials are no usual vector spaces, as polynomials are elements of rings instead of fields. The resulting objects are called modules.

2.4.1 Modules

Modules are a generalisation of vector spaces for rings. If their elements are vector–like elements of R^n, they will still be called vectors within this thesis. This section will revisit some known properties of modules from [Art91, Chapter 12].

Definition 2.4.1 (Left Module)
Let R be a, not necessarily commutative, ring with one. A left R–module consists of some commutative additive group M and an operation, with $a, b \in R$ and $v, u \in M$, satisfying:

$$\begin{aligned} 1v &= v \\ (ab)v &= a(bv) \\ (a+b)v &= av + bv \\ a(v+u) &= av + au. \end{aligned}$$

This definition is analogue to vector spaces. Some important difference between vector spaces and modules is the non existence of a basis for most modules. Modules with a basis are called *free modules*. For modules isomorphic to a vector–like module of the form R^n the Theorem 2.4.1 follows.

Theorem 2.4.1
Let R be some, not necessarily commutative, ring. For any $n \in \mathbb{N}$ the module R^n is a free (left- or right-) module.

Proof. Let 1 be the one element of R. A basis B of R^n is then $B = \{b_i : 1 \leq i \leq n\}$ with b_i zero everywhere but position i, where it is one:

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

□

From now on, the considered module is some module R^n over polynomials, either regular or q-polynomials. This module is a free module and has a basis.

2.4.2 Module Minimisation

Module minimisation is the method of finding a reduced basis for a module constituting of the shortest vectors of the module. For polynomial modules the shortest vectors are considered based on their degree.

Definition 2.4.2 (Degrees of Vectors and Matrices)
Let $\mathbf{v} \in \mathbb{F}[x;\theta;\delta]^n$ be a polynomial vector with $\mathbf{v}_i$ the i-th position in $\mathbf{v}$. Let further $\mathbf{M} \in \mathbb{F}[x;\theta;\delta]^{l\times n}$ be a polynomial matrix with m_i the i-th row of $\mathbf{M}$. Their respective degrees are then:

$$\deg \mathbf{v} = \max\{\deg \mathbf{v}_i : 1 \leq i \leq n\},$$
$$\deg \mathbf{M} = \sum_{i=0}^{l} \deg m_i.$$

A useful measurement for polynomial vectors is the leading position, which connects the degree of a vector with its degree defining elements.

Definition 2.4.3 (Leading Position)
The leading position LP of a vector $v \in \mathbb{F}[x]^n$ is defined by

$$LP(v) := \max\{i : \deg v = \deg v_i\}.$$

Informally, the leading position in a row vector is the polynomial of highest degree. If this is ambiguous, the right most position of those highest degree polynomials is used. Using this definition of a leading position, [MS03] defines the weak Popov form.

Definition 2.4.4 (Weak Popov Form)
A matrix $\mathbf{M} \in \mathbb{F}[x]^{n\times m}$ is in weak Popov form, if all leading positions of non–zero rows are different.

Example 2.4.1 (Weak Popov Form and Leading Positions)
The matrix A is not in weak Popov form. This can be seen in the degree matrix B with $b_{i,j} = \deg a_{i,j}$, a matrix containing the degrees of corresponding polynomials of A. Leading positions and their corresponding degree are marked. There is a conflict between row one and row three.

$$A = \begin{pmatrix} x^2+1 & x^2 & \boxed{x^2+x} \\ \boxed{x^5} & x^2+x & x+1 \\ 0 & 0 & \boxed{1} \end{pmatrix}, B = \begin{pmatrix} 2 & 2 & \boxed{2} \\ \boxed{5} & 2 & 1 \\ -\infty & -\infty & \boxed{0} \end{pmatrix}$$

A basis in weak Popov form, following [Ale05, Proposition 2.3], will result in the shortest vector being part of the basis.

Theorem 2.4.2
Let $\mathbf{M} \in \mathbb{F}[x;\theta;\delta]^{l\times n}$ be some basis in matrix form in weak Popov form. Any non–zero row $\mathbf{m}$ of $\mathbf{M}$ with minimal degree is the shortest vector.

Proof. Assume a vector

$$\mathbf{v} = \sum_{i=0}^{l} c_i \mathbf{m}_i \tag{2.7}$$

with $\deg \mathbf{v} < \deg \mathbf{m}$. Consider row $\mathbf{m}_i$ with $\deg \mathbf{m}_i$ maximal within $\mathbf{M}$ and $c_i \neq 0$. If there are several with this quality, consider the vector $\mathbf{m}_i$ of those with maximum degree, with maximum leading position. As $\mathbf{M}$ is in weak Popov form, this is uniquely possible, as all leading positions are distinct.

The factor $c_i\mathbf{m}_i$ has to cancel out in the sum of Equation (2.7). For this to happen, another row $\mathbf{m}_j, j \neq i$ is needed. But as $\mathbf{m}_i$ was the unique vector with maximum degree at this position, a factor $c_j\mathbf{m}_j$ would increase or keep the degree as $\deg c_j\mathbf{m}_j \geq \deg c_i\mathbf{m}_i$ at a different position. □

To utilise this method, an algorithm is needed to transform a given basis into weak Popov form.

Theorem 2.4.3
There exists at least one algorithm to transform an arbitrary basis into a basis in weak Popov form.

An algorithm is given by [MS03] for usual polynomials. For Ore extension polynomials [LNPS15] gives two algorithms. This method will be utilised in the following sections on Reed–Solomon and Gabidulin codes to find a shortest vector solution within a space of solutions.

2.5 Determinant and Orthogonality Defect

For a more detailed complexity analysis of the algorithms a concept of determinant of matrices over non–commutative rings is useful, as it allows further restrictions. But the determinant is only defined over fraction fields. This section will revisit already known results, that show the existence of a determinant and some useful

properties, as well as one important conclusion. For a more explicit version confer [LNPS15, Section 4.1].

2.5.1 Existence of a Determinant

The determinant will be defined over a fraction field, so the Ore extension needs to be extended to some fraction field. [Coh95] examines Ore extensions and skew polynomials in Section 1.3 and 2.1.

In [Coh95, Section 1.3], it is shown that left Ore principal ideal domain can be embedded in a fraction field $\mathcal{Q}$. As mentioned in Section 2.3, $\mathbb{F}[x;\theta;\delta]$ fulfils these conditions for $\delta = 0$ and the Frobenius automorphism $\theta(a) = a^q$.

The important measurement of degrees has to be extended to the fraction fields, as it will be the main evaluation criteria for the determinant.

Definition 2.5.1 (Degree in Fraction Field [LNPS15])
The degree of a ring R can be easily extended to the fraction field $\mathcal{Q}$ by the formula:

$$\deg(g^{-1}f) = \deg f - \deg g.$$

Using the existence of an embodiment of some Ore extension fulfilling the conditions, the following theorem, a part of [Ros94, Theorem 2.2.5], introduces the determinant and some important properties. The determinant used is called the Dieudonné determinant.

Lemma 2.5.1
Let $I, A, B \in \mathcal{Q}^{n\times n}$ be some quadratic matrices over a not necessarily commutative fraction field over a left Ore ring R and $a, b \in \mathcal{Q}$ elements of the fraction field of R. Further let ϕ be a function specified by [Ros94, Theorem 2.2.5 (c)] with the property $\deg\phi(f) = \deg f$. There exists a function $det : \mathcal{Q}^{n\times n} \longrightarrow \mathcal{Q}$ having the following properties:

1. $\det I = 1$, *where I is the identity matrix.*
2. *If a matrix B is created from some matrix A by adding a left multiple of one row to another row, then* $\det B = \det A$.
3. *If a matrix B is created from some matrix A by left multiplying one of the rows by a, then* $\det B = \phi(a)\det A$.
4. *If a matrix B is created from some matrix A by interchanging two rows, then* $\det B = \phi(-1)\det A$.

Lemma 2.5.1(4) follows, that the degree of the determinant is invariant regarding row interchanging, as $\deg \phi(-1) = 0$.

2.5.2 Orthogonality Defect

Based on the determinant the orthogonality defect can be defined.

Definition 2.5.2 (Orthogonality Defect [LNPS15])
The orthogonality defect is defined by

$$\Delta\mathbf{M} = \deg \mathbf{M} - \deg \det \mathbf{M}.$$

There is an important connection between the orthogonality defect and the weak Popov form of a matrix described by [Nie13a] and applied for the Dieudonné determinant by [LNPS15, Lemma 7] which is restated in the following Theorem 2.5.1.

Theorem 2.5.1
Let $\mathbf{M} \in \mathcal{Q}^{n\times n}$, with $\mathcal{Q}$ being the fraction field of some left Ore ring, be some matrix in weak Popov form. It then holds, that:

$$\Delta\mathbf{M} = 0.$$

Proof. As $\mathbf{M}$ is over the fraction field, left inverse elements exist.

Lemma 2.5.1(4) allows interchanging of rows, so that the leading positions are on the diagonal, without interfering with the degree of the determinant.

Using Lemma 2.5.1(2), the matrix can be transformed to upper triangular form using elementary row operations. This is possible, due to the matrix being defined over a fraction field and the existence of left inverses. As the leading positions are all different, the product to zero out an entry has a degree lower than the leading positions right of the target row's leading position and as the leading position.

Up to this point, the degrees of the leading positions are preserved and they are still on the diagonal.

The matrix is now in upper triangular form with leading positions still on the diagonal. Starting from the bottom using elementary row operations, this matrix can be transformed into a diagonal matrix. If a row is used for an elementary row operation, every entry besides the diagonal element is zero. This will preserve the degree of the diagonal elements and the determinant, as it uses Lemma 2.5.1(2).

The diagonal matrix can be created from the identity matrix I by multiplying every row with the corresponding diagonal element. As Lemma 2.5.1(1) states, the identity matrix has $\deg\det I = 1$. Lemma 2.5.1(3) allows multiplying a row by an element and furthermore the determinant can be calculated by $\det B = \phi(a)\det A$.

Repeated application of this results in $\det\mathbf{U} = (\prod_{i=1}^{n}\phi(\mathbf{U}_{i,i}))\det\mathbf{I}$. Applying the degree function will result in:

$$\begin{aligned}\deg\det\mathbf{M} &= \deg((\prod_{i=1}^{n}\phi(\mathbf{M}_{i,i}))\underbrace{\det\mathbf{I}}_{=1})\\ &= \deg(\prod_{i=1}^{n}\phi(\mathbf{M}_{i,i})) + \underbrace{\deg\det\mathbf{I}}_{=0}\\ &= \sum_{i=1}^{n}\deg\phi(\mathbf{M}_{i,i}).\end{aligned}$$

As the degree of the diagonal elements is equivalent to the degree of the leading positions in the beginning, it is:

$$\deg\det\mathbf{M} = \sum_{i=1}^{n}\deg\phi(\mathbf{M}_{i,i}) = \sum_{i=1}^{n}\deg LP(\mathbf{M}_i) = \sum_{i=1}^{n}\deg\mathbf{M}_i = \deg\mathbf{M}.$$

The theorem follows. □

Another conclusion of the final steps is the following lemma.

Lemma 2.5.2
A matrix $\mathbf{M}$ *in upper triangular form has a degree of its determinant*

$$\deg\det\mathbf{M} = \sum_{i=1}^{n}\deg\mathbf{M}_i.$$

This theorem and lemma and the orthogonality defect will prove useful during complexity analysis in Chapter 4.

3 Coding Theory Fundamentals

The algorithms examined in this thesis are evaluated for decoding of Gabidulin codes, which are introduced in Section 3.2. As they are the rank metric equivalent of the wide spread Reed–Solomon codes, Reed–Solomon codes are introduced in Section 3.1, too.

3.1 Reed–Solomon Codes

Reed–Solomon codes are widely used in many products and specifications. They are used for QR–codes [ISO06], audio discs, the voyager space probes [Wic94] and many more. Within this thesis they are used due to their similarities to Gabidulin codes. Also, the decoding algorithm of interest stems from Reed–Solomon codes and has been applied to Gabidulin codes recently.

Reed–Solomon codes are widely studied and part of many books on coding theory. A source of information on them is [Rot06].

Definition 3.1.1 (Generalised Reed–Solomon Code [Nie13b])
Let $\mathbb{F}$ be a finite field, the $[n, k, d]$ Generalised Reed–Solomon (GRS) code is the set

$$\mathcal{C}_{GRS} = \{(\beta_1 f(\alpha_1), \ldots, \beta_n f(\alpha_n)) : \begin{array}{l} f \in \mathbb{F}[x] \text{ with } \deg f < k, \\ \beta_j \in \mathbb{F}\backslash\{0\}, \alpha_i \in \mathbb{F}, \forall i \neq j : \alpha_i \neq \alpha_j \end{array} \Big\}.$$

For some decoding algorithms $a_i \neq 0$ is necessary (cf. [Nie13b]). This definition is the basis for the further operations.

3.1.1 Hamming Metric

Distance measure for Reed–Solomon codes is done using the Hamming metric, which is defined via the Hamming weight.

Definition 3.1.2 (Hamming Weight and Hamming Distance [Ham50])
Let $\mathbf{a}, \mathbf{b} \in \mathcal{G}^n$ *be vectors of length* n *containing elements of some additive group* $(\mathcal{G}, +, 0)$*, then the Hamming weight* $\mathrm{wt}(\mathbf{a})$ *and distance* $\mathrm{dist}(\mathbf{a}, \mathbf{b})$ *are defined by:*

$$\begin{aligned} \mathrm{wt}(\mathbf{a}) &= |\{i : a_i \neq 0\}|, \\ \mathrm{dist}(\mathbf{a}, \mathbf{b}) &= \mathrm{wt}(\mathbf{a} - \mathbf{b}). \end{aligned}$$

Informally, the Hamming weight is the number of entries of a vector that are not equal to zero. The Hamming distance is the number of entries that differ between two vectors.

3.1.2 Encoding of Reed–Solomon Codes

Encoding of Reed–Solomon codes is the method of associating a polynomial f to some information v, where f will be used as in Definition 3.1.1. There are multiple encoding schemes for Reed–Solomon codes, including systematic and non–systematic schemes. [Bos13, Section 3.1.5] lists four methods using a generator- and test–polynomial, including the following one.

Definition 3.1.3 (Associated Polynomial)
Let $v \in \mathbb{F}_q^k$ *be some information. Then the associated polynomial* f *of* i *is*

$$f(x) = \sum_{i=0}^{k-1} v_{i+1} x^i = v_1 + v_2 x + v_3 x^2 + \ldots + v_k x^{k-1}.$$

The k positions of the information are used as coefficients of a polynomial with a degree strictly smaller than k, the condition of Definition 3.1.1.

Example 3.1.1 (Reed–Solomon Encoding)
For a continuous example of a $[5, 3, 3]$ *GRS code over* $\mathbb{F}_7$*, set the parameters to*

$$\begin{aligned} &\beta_1 = \beta_2 = \beta_3 = \beta_4 = \beta_5 = 1, \\ &\alpha_1 = 3, \\ &\alpha_2 = \alpha_1^2 = 2, \\ &\alpha_3 = \alpha_1^3 = 6, \\ &\alpha_4 = \alpha_1^4 = 4, \\ &\alpha_5 = \alpha_1^5 = 5. \end{aligned}$$

Let $v = (5, 2, 3) \in \mathbb{F}_7^3 = \mathbb{F}^k$ be the information to encode. We associate the polynomial

$$\begin{aligned} f(x) &= \sum_{i=0}^{k-1} v_{i+1} x^i \\ &= v_1 + v_2 x + v_3 x^2 + \ldots + v_k x^{k-1} \\ &= 5 + 2x + 3x^2 \end{aligned}$$

with v and compute the code word $c \in \mathbb{F}_7^5 = \mathbb{F}^n$ as

$$\begin{aligned} c &= (\beta_1 f(\alpha_1), \beta_2 f(\alpha_2), \beta_3 f(\alpha_3), \beta_4 f(\alpha_4), \beta_5 f(\alpha_5)) \\ &= (f(3), f(2), f(6), f(4), f(5)) \\ &= (3, 0, 6, 5, 6). \end{aligned}$$

This example will be continued in the following step of decoding Reed–Solomon codes.

3.1.3 Decoding of Reed–Solomon Codes

For the decoding process we consider some unknown codeword $c \in \mathcal{C}_{GRS}(n, k)$, and some unknown error $e \in \mathbb{F}^n$, which is hoped to be mostly zero. The known received word is constructed by $r = c + e$. For generalised Reed–Solomon codes with $\beta_i \neq 0$ we set $r' = (\frac{r_1}{\beta_1}, \ldots, \frac{r_n}{\beta_n})$ and use r' instead of r.

A widely used concept, the error locator polynomial, is elementary to many decoding algorithms, including the one examined in this section. But not everyone uses the same definition. The one that will be used in this thesis is the error locator polynomial having roots in all error positions.

Definition 3.1.4 (Error Locator Polynomial [Nie13a])
The error locator polynomial is defined as

$$\Lambda(x) = \prod_{i \in \mathcal{E}} (x - \alpha_i).$$

As the codewords are confined by the code locators and the degree limit, equivalences are usually modulo some polynomial. It is useful to have some simplification to prove equivalence modulo some polynomial.

Theorem 3.1.1
Two polynomials $f, h \in \mathbb{F}[x]$ are equivalent modulo some polynomial $G \in \mathbb{F}[x]$ if and only if they evaluate the same for all roots of G:

$$(\forall \alpha : G(\alpha) = 0) \Longrightarrow f(\alpha) = h(\alpha).$$

Proof. It is equivalent to write

$$f \equiv h \mod G \tag{3.1}$$
$$\Leftrightarrow\ G | f - h. \tag{3.2}$$

Choose some α such that it is a root of G. As α is a root, some factor $(x - \alpha)$ divides G. As $(x - \alpha)|G$ is true, for Equation (3.2) to be true, it must also be true that $(x - \alpha)|f - h$, which is equivalent to α being a root of $f - h$. But for α to be a root of $f - h$ it has to be that $f(\alpha) = h(\alpha)$. Since α was an arbitrary root of G, all roots of G have to be roots of $f - h$. □

Combining the above, [Gao03] concluded a relationship of the error locator polynomial, the interpolation of the received word and the information polynomial modulo the polynomial with the code locators as roots.

Definition 3.1.5 (Gao Key Equation [Gao03])
Let λ be some solution for Λ, ψ a solution for $\Omega = (\Lambda f)$, $r(x)$ the interpolation polynomial of the received word r with $r(\alpha_i) = r_i$, and $G(x) = \prod_{i=1}^{n}(x - \alpha_i)$ the minimal polynomial with roots in all code locators. Then the Gao key equation is

$$\Lambda(x)r(x) \equiv \Omega \mod G(x), \tag{3.3}$$

with additional requirements for a solution λ, ψ:

$$\deg \lambda + (k - 1) \geq \deg \psi. \tag{3.4}$$

The right hand side of Equation (3.3) is an abstraction of its structure with $\Omega = \Lambda \cdot f$ as a substitute. The original equation would be $\Lambda(x)r(x) \equiv \Lambda(x)f(x) \mod G(x)$.

Proof. Using Theorem 3.1.1 it is sufficient to show, that $\Lambda(x)f(x)$ and $\Lambda(x)r(x)$ evaluate the same for all roots of $G(x)$, which are the code locators.

For $e_i \neq 0$ it is $\Lambda(i) = 0$, so both sides are zero. For $e_i = 0$ it is $r(i) = c_i = f(i)$. □

To get a solution (λ, ψ) for Equation (3.3) let $\lambda = a$:

$$\begin{aligned} & \lambda r = \psi + d \cdot G \\ \Leftrightarrow \quad & \psi = \lambda \cdot r + c \cdot G \qquad (\text{with } c = -d) \\ \Rightarrow \quad & (\lambda,\ \psi) = (a,\ a \cdot r + c \cdot G) \\ & \qquad = a(1,\ r) + c(0,\ G). \end{aligned}$$

The result is a submodule solution space with row basis $\beta = \{(1, r), (0, G)\}$ for Equation (3.3), but not every solution within this submodule fulfils the degree requirements of Equation (3.4).

To restrict the space of solutions, a solution should not only satisfy Equation (3.4) but also be minimal with respect to $\deg \lambda + (k-1)$. Therefore, the solution space has to be shifted, which is equivalent to a multiplication with a diagonal matrix having the shifts as diagonal elements:

$$\mathbf{D} = \begin{pmatrix} x^{k-1} & 0 \\ 0 & 1 \end{pmatrix}.$$

The solution of interest is the shortest vector of the resulting submodule, that still fulfils the side condition. This would not always be the correct solution, as Λ does not have to be the minimal solution, in those cases decoding would fail. However, this only occurs for $|\{i : e_i \neq 0\}| \geq d/2$ (cf. [Nie13b], as non interleaved decoding is a special case of virtually interleaved decoding).

The solution can be computed using the algorithm of [Nie13a] without powering: Let $\mathbf{M}$ be a basis for the solution and $\mathbf{D}$ a diagonal shift matrix of the form above. Compute $\mathbf{M}'$ such that $\mathbf{M}'\mathbf{D}$ is in weak Popov form. Check the basis vectors of M' for a valid solution. If such a solution is found, return it, otherwise declare decoding failure.

Example 3.1.3 can be considered for simple decoding, too, but is used in context of virtually interleaved codes.

3.1.4 Interleaved Reed–Solomon Codes

Interleaved Reed–Solomon codes are used in several instances as compact discs or other data storage, transmission or processing units (cf. [SSB09]). Gabidulin codes can be interleaved as well and show many similarities to interleaved Reed–Solomon codes.

The following definition is the basis for operations using interleaved Reed–Solomon codes.

Definition 3.1.6 (Interleaved Reed–Solomon Codes [SSB09])
Let $C_1, \ldots, C_\ell$ be, not necessarily distinct, Reed–Solomon codes of length n and dimensions $k_1, \ldots, k_s$. The interleaved Reed–Solomon code $IRS[\ell; n, k_1, \ldots, k_\ell]$ is defined by:

$$IRS[\ell; n, k_1, \ldots, k_s] = \{\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{pmatrix} : c_i \in C_i, i \in 1 \ldots \ell\}.$$

The process of encoding is simply encoding of ℓ regular Reed–Solomon codes. it is possible to decode every single codeword separately, but for collaborative decoding, a new key equation is necessary.

Definition 3.1.7 (Key Equation [Nie13a])
Let λ be a solution for Λ, ψ_t a solution for (Λf_t) and $G(x) = \prod_{i=1}^{n}(x - \alpha_i)$ the minimal polynomial with roots in all code locators. The key equation is

$$\Lambda(x) r_t(x) \equiv \Lambda(x) f_t(x) \mod G(x), \tag{3.5}$$

with additional requirements for a solution $\lambda, \psi_1, \ldots, \psi_\ell$:

$$\deg \lambda + (k_i - 1) \geq \deg \psi_i.$$

The process to get a solution is analogue to the process in Section 3.1.3. The matrix is simple a union of the single solution matrices:

$$\mathbf{M} = \begin{pmatrix} 1 & r_1 & r_2 & \ldots & r_\ell \\ 0 & G_1 & 0 & \ldots & 0 \\ 0 & 0 & G_2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & G_\ell \end{pmatrix}.$$

But another shift matrix is needed, as more dependencies have to be considered. This results in the following diagonal matrix, where $\gamma_i = \sum_{j=0}^{\ell}(k_j - 1) - (k_i - 1)$ and $\gamma_\lambda = \sum_{j=0}^{\ell}(k_j - 1)$ are the shifts for the degree requirements:

$$\mathbf{D} = \mathrm{diag}(x^{\gamma_\lambda}, x^{\gamma_1}, \ldots, x^{\gamma_\ell}).$$

All that is left is to retrieve the original information.

Retrieving the Information Polynomial

One advantage of Gao style decoding is the direct computation of the information polynomial, instead of needing further steps like the Forney formula. For some valid solution $(\lambda, \psi_1, \psi_2, \ldots, \psi_l)$, the information polynomial f is

$$f_i = \frac{\lambda}{\psi_i},$$

because the right hand side of the Gao key equation (cf. Equation (3.3)) had the structure $\psi = \Lambda f$.

Example 3.1.2 (Final Decoding Step)
Using the resulting polynomials of Example 3.1.3, the information polynomial f is easily computable:

$$\begin{aligned}
\lambda(x) &= 5x + 1 && = (x+3) \cdot 5 \\
\psi_1(x) &= x^3 + 6x^2 + 6x + 5 && = (x+5)^2 \cdot (x+3) \\
f(x) &= \frac{\psi_1(x)}{\lambda(x)} && = (x+5) \cdot (3x+1) = 3x^2 + 2x + 5.
\end{aligned}$$

This is the information polynomial computed in Example 3.1.1 and has the original information as coefficients.

3.1.5 Application of Interleaved Reed–Solomon Codes

Virtually interleaving of Reed–Solomon codes, called power decoding, is one method to decode beyond half the minimum distance. This technique was introduced by [SSB06]. One possibility to virtually interleave a given code is to create new received words by powering the original received word r:

$$\begin{aligned}
r_t^n &= (c_t + e_t)^n \\
&= \sum_{i=0}^{n} \binom{n}{k} c_t^{n-i} e_t^i \\
&= c_t^n + \underbrace{\sum_{i=1}^{n} \binom{n}{k} c_t^{n-i} e_t^i}_{=\tilde{e_t},n}.
\end{aligned}$$

The error occurring for those words is $\tilde{e}_{t,n}$. An important observation is $e_t = 0 \Longrightarrow \tilde{e}_{t,n} = 0 \Longleftrightarrow r_t^n = c_t^n$. Whenever the error is zero for the original word, it is zero for the powered word, too. Another consequence of this is one resulting Λ for all powered words.

Definition 3.1.8 (Degree Requirements [Nie13a])
Virtually interleaved Reed–Solomon codes allow for simpler degree requirements, as it holds, that $\forall i, j \; k_i = k_j$.

$$\deg \lambda + t(k-1) \geq \deg \psi_t \tag{3.6}$$
$$t(k-1) < n. \tag{3.7}$$

This results in the shift matrix:

$$\mathbf{D} = \operatorname{diag}(x^{l(k-1)+1}, x^{(l-1)(k-1)}, x^{(l-2)(k-1)}, \ldots, x^{k-1}, 1).$$

The application of this algorithm is shown in Example 3.1.3.

Example 3.1.3 (Module Minimisation to Solve for Λ)
Let $r = (3, 0, 6, 1, 6)$ *be the received word from Example 3.1.1. At first we compute* $G(x)$, R_1 *and* R_2*:*

$$\begin{aligned} G(x) &= \prod_{i=1}^{n}(x - \alpha_i) \\ &= (x-3)(x-2)(x-6)(x-4)(x-5) \\ &= x^5 + x^4 + x^3 + x^2 + x + 1 \\ R_1(x) &= 6x^4 + 2x^3 + 3x^2 + x \\ R_2(x) &= 3x^4 + 3x^3 + 6x^2 + 2. \end{aligned}$$

This results in a basis $\mathbf{M}$ *and a shift matrix* $\mathbf{D}$*.*

$$\mathbf{M} = \begin{pmatrix} 1 & R_1 & R_2 \\ 0 & G & 0 \\ 0 & 0 & G \end{pmatrix}, \mathbf{D} = \begin{pmatrix} x^5 & 0 & 0 \\ 0 & x^2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

We now compute $\mathbf{M}'$*, a basis of* $\mathbf{M}$*, such that* $\mathbf{M}'\mathbf{D}$ *is in weak Popov form. A possible result is:*

$$\mathbf{M}' = \begin{pmatrix} 5x+1 & x^3 + 6x^2 + 6x + 5 & x^5 + 4x^4 + 5x^3 + 6x^2 + 3x + 2 \\ x & 3x^4 + 4x^3 + 2x^2 + x + 1 & 3x^5 + 3x^4 + 6x^3 + 2x \\ 0 & 0 & x^5 + x^4 + x^3 + x^2 + x + 1 \end{pmatrix}.$$

Only the first row of M' fulfils the requirement of Equation (3.6) *of the power Gao key equations, so we assume that row is a solution for* Λ *in the power Gao equations of Equation* (3.3)*. This results in the following solutions to* Λ *and* Ω*.*

$$\begin{aligned} \lambda(x) &= 5x + 1 \\ \psi_1(x) &= x^3 + 6x^2 + 6x + 5 \\ \psi_2(x) &= x^5 + 4x^4 + 5x^3 + 6x^2 + 3x + 2 \end{aligned}$$

Finding the roots of λ can be done by exhaustive search, resulting in:

$$\lambda(x) = 0 \Longleftrightarrow x = 4 = \alpha_4.$$

Hence the error is on position four: $(3, 0, 6, \boxed{1}, 6)$. *In comparison to Example 3.1.1:* $(3, 0, 6, 5, 6)$.

3.2 Gabidulin Codes

This section introduces Gabidulin codes, the rank metric equivalent of Reed–Solomon codes of Section 3.1. This equivalence enables conversion of concepts from Reed–Solomon codes to Gabidulin codes, an important source of research on Gabidulin codes. The algorithm of this thesis was first used for Reed–Solomon codes as well.

3.2.1 Rank Metric

The basis for distance measuring of Gabidulin codes is different from Reed–Solomon codes Hamming metric. For calculating the distance, we use the rank of the matrix, which was transformed throguh the ext mapping from Section 2.1.2.

Definition 3.2.1 (Rank Metric [Gab85])
Let $a, b \in \mathbb{F}_{q^m}$ be two elements of some extension field $\mathbb{F}_{q^m}$. The rank weight and distance are defined by the equations

$$\begin{aligned} \mathrm{wt}_{rk}(\mathbf{a}) &= \mathrm{rank}\left(\mathrm{ext}\left(\mathbf{a}\right)\right) = \mathrm{rank}\left(\mathbf{A}\right), \\ \mathrm{dist}_{rk}\left(\mathbf{a}, \mathbf{b}\right) = \mathrm{wt}_{rk}(\mathbf{a} - \mathbf{b}) \quad &= \mathrm{rank}\left(\mathrm{ext}\left(\mathbf{a}\right) - \mathrm{ext}\left(\mathbf{b}\right)\right) = \mathrm{rank}\left(\mathbf{A} - \mathbf{B}\right). \end{aligned}$$

3.2.2 Encoding of Gabidulin Codes

The definition of Gabidulin codes is similar to the definition of Reed–Solomon codes.

Definition 3.2.2 (Gabidulin Codes [WZ13])
Let $g_1, \ldots, g_n \in \mathbb{F}_{q^m}$ be linear independent over $\mathbb{F}_q$ and $f \in \mathcal{L}_q[x]$ with $\deg_q f < k$. Then the Gabidulin code is the set

$$\mathcal{C}_{Gab} = \{(f(g_1), f(g_2), \ldots, f(g_n))\}.$$

The elements $g_1, \ldots, g_n$ will be called code locators of $\mathcal{C}_{Gab}$. The function f is a q-polynomial associated with the information to encode, similar to encoding of Reed–Solomon codes in Section 3.1.2.

While encoding some information as coefficients of f is still possible, [KK08] suggests using subspaces as information. As linearised polynomials have subspaces as root spaces, the association can be defined by this.

3.2.3 Decoding of Gabidulin Codes

There are multiple approaches to decoding for Gabidulin codes. For this thesis, the relevant approach uses a key equation similar to the Gao key equation of Reed–Solomon codes from Definition 3.1.5. For such an equation multiple concepts need to be redefined or introduced for Gabidulin codes and non commutative polynomials.

Preparations

Lemma 3.2.1
Let $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ be a matrix of rank $t = \operatorname{rank}(\mathbf{A})$. There exist full rank matrices $\mathbf{B} \in \mathbb{F}_q^{m \times t}$, $\mathbf{C} \in \mathbb{F}_q^{t \times n}$ with row space $\mathcal{R}$ and column space $\mathcal{C}$ that fulfil:

$$\mathcal{R}(\mathbf{A}) = \mathcal{R}(\mathbf{B}),$$
$$\mathcal{C}(\mathbf{A}) = \mathcal{C}(\mathbf{C}).$$

Lemma 3.2.1 goes back to and has been proven by [MS74, Theorem 1]. The decomposition of two matrices will be used to model the error of Gabidulin code transmissions. Consider some received word $\mathbf{r} = \mathbf{c} + \mathbf{e}$ and its matrix representation $\operatorname{ext}(\mathbf{r}) = \mathbf{R} = \mathbf{C} + \mathbf{E}$ with $\operatorname{rank}(\mathbf{E}) = t$.

Lemma 3.2.1 allows for a full rank decomposition of the error matrix $\operatorname{ext}\mathbf{e} = \mathbf{E} = \mathbf{A} \cdot \mathbf{B}$, where $\mathbf{A} \in \mathbb{F}_q^{m \times t}$ and $\mathbf{B} \in \mathbb{F}_q^{t \times n}$ and $\operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{B}) = t$.

Let $\mathbf{a} = \operatorname{ext}^{-1}\mathbf{A}$ be the ext inverse of the left part of the error decomposition. This allows for the following decomposition:

$$\mathbf{e} = \operatorname{ext}^{-1}(\mathbf{E}) \overset{Eq.\ 2.2}{=} \operatorname{ext}^{-1}(\mathbf{A}) \cdot \mathbf{B} = \mathbf{a} \cdot \mathbf{B}. \tag{3.8}$$

Based on this definition the error span polynomial will be used for decoding, similar to the error locator polynomial of Definition 3.1.4 of Reed–Solomon codes.

Definition 3.2.3 (Error Span Polynomial [WZ13])
Let $\mathbf{a} = (a_1, a_2, \ldots, a_t)$ *be the left part of the error decomposition of Equation* (3.8). *The error span polynomial is defined as*

$$\Lambda(x) = M_{\langle \mathbf{a} \rangle} = M_{\langle a_1, a_2, \ldots, a_t \rangle} = \prod_{\alpha \in \langle \mathbf{a} \rangle} (x - \alpha).$$

At last some interpolation of the received word is necessary, as a regular interpolation would result in a regular polynomial.

Definition 3.2.4 (Interpolation [WZ13])
Let $g_i \in \mathcal{G}$ *be the code locators of some Gabidulin code. Let now* $L_i(x) = M_{\mathcal{G} \setminus g_i} \in \mathcal{L}_q[x]$ *be a linearised helper polynomial. The interpolation* $\hat{r}$ *of some word* $r \in \mathbb{F}_{q^m}^n$ *with* $r = (r_1, r_2, \ldots, r_n)$ *is then*

$$\hat{r} = \sum_{i=1}^{n} r_i \frac{L_i(x)}{L_i(g_i)}.$$

One useful observation is $\hat{r}(g_i) = r_i$, as

$$\frac{L_i(g_j)}{L_i(g_i)} = \begin{cases} 1 & if\ i = j, \\ 0 & else. \end{cases}$$

This is true, as g_j is a root of L_i, but g_i is not, therefore it is either $L_i(g_i)$ divided by itself, or zero divided by something non–zero.

The following theorem is equivalent to [WZ13, Theorem 3.6] but this proof is slightly different. It combines the concepts into an equation similar to the Gao key equation of Definition 3.1.5.

Theorem 3.2.1
Let $\hat{r}$ *be the interpolation of* r *using Definition 3.2.4. Let* f *be the unknown information polynomial used to compute the codeword* c, $M_{\mathcal{G}}$ *the minimal subspace polynomial over the code locators and* Λ *the error span polynomial. Then the Gao–like key equation for Gabidulin codes is*

$$\Lambda(\hat{r}(x) - f(x)) \equiv 0 \mod M_{\mathcal{G}}. \tag{3.9}$$

Proof. Let $G_i \in \mathbb{F}_q$ be arbitrary constants for $1 \leq i \leq n$ and g_i are the code locators of some Gabidulin code. Let $\mathbf{r}$ be the received word, then $\hat{r}$ is the unique q-polynomial fulfilling $\hat{r}(g_i) = \mathbf{r}_i$ and $\deg_q(\hat{r}) < n$. f shall be the information q-polynomial used to compute the codeword $\mathbf{c}$ that led to receiving $\mathbf{r}$. Now evaluate

Equation (3.9) with $x = \sum_{i=1}^{n} G_i g_i$:

$$\begin{aligned}
\Lambda(\hat{r}(\sum_{i=1}^{n} G_i g_i) - f(\sum_{i=1}^{n} G_i g_i)) &\overset{\mathbb{F}_q \text{ linear}}{=} \Lambda(\sum_{i=1}^{n} G_i \hat{r}(g_i) - \sum_{i=1}^{n} G_i f(g_i)) \\
&= \sum_{i=1}^{n} G_i \Lambda(\hat{r}(g_i) - f(g_i)) \\
&= \sum_{i=1}^{n} G_i \Lambda(r_i - c_i) \\
&= \sum_{i=1}^{n} G_i \underbrace{\Lambda(e_i)}_{=0} \\
&= 0.
\end{aligned}$$

As all G_i have been arbitrary Equation (3.9) evaluates to zero on both sides for all $\mathbb{F}_q$ linear combinations of the code locators g_i. Those linear combinations are the roots of the minimal subspace polynomial $M_{\mathcal{G}} = M_{\langle g_i \rangle}$. Therefore both sides evaluate the same for all roots of the minimal subspace polynomial $M_{\mathcal{G}}$. The requirements of Theorem 2.2.1 are therefore fulfilled. The theorem follows. □

Theorem 3.2.2
Let $\hat{r}$ be the interpolation of r using Definition 3.2.4. Let f be the unknown information polynomial used to compute the codeword c and $M_{\mathcal{G}}$ the minimal subspace polynomial over the code locators. Then a valid transformation of Theorem 3.2.1 is

$$\Lambda(\hat{r}(x)) \equiv \Lambda(f(x)) \mod M_{\mathcal{G}}. \tag{3.10}$$

Let λ be some solution for Λ and ψ a solution for $\Omega = (\Lambda(f))$. The degree restriction for solutions is then formulated by

$$\deg_q \lambda + (k-1) \geq \deg_q \psi. \tag{3.11}$$

Proof. Again, it is sufficient to show that both sides evaluate the same for every g_i. For i with $e_i = 0$ it is $\Lambda(f(g_i)) = \Lambda(c_i) = \Lambda(r_i) = \Lambda(\hat{r}(g_i))$. For i with $e_i \neq 0$ it

holds, that:

$$\begin{aligned}\Lambda(\hat{r}(g_i)) &= \Lambda(r_i)\\ &= \Lambda(c_i + e_i)\\ &= \Lambda(c_i + \sum_{j=1}^{t} B_{j,i}a_j)\\ &\overset{Eq\ 2.2}{=} \Lambda(c_i) + \underbrace{\sum_{j=1}^{t} B_{j,i}\Lambda(a_j)}_{=0}\\ &= \Lambda(c_i)\\ &= \Lambda(f(g_i)).\end{aligned}$$

□

The transformed equation can be solved using module minimisation.

Module Minimisation

[LNPS15, Lemma 1] gives the solution space for interleaved codes, where the shifts for the degrees are described by the function

$$\Phi : \mathcal{L}_q[x]^\ell \longrightarrow \mathcal{L}_q[x]^\ell : (u_1, \ldots, u_\ell) \longrightarrow (u_1 x^{[\gamma_1]}, \ldots, u_\ell x^{[\gamma_\ell]}),$$

where $\gamma_1, \ldots, \gamma_\ell$ are the weights. The solution space can be reduced to the non–interleaved version using the matrices

$$\mathbf{B} = \begin{pmatrix} 1 & \hat{r} \\ 0 & M_\mathcal{G} \end{pmatrix}, \Phi(\mathbf{B}) = \begin{pmatrix} x^{[k-1]} & \hat{r} \\ 0 & M_\mathcal{G} \end{pmatrix}.$$

Theorem 3.2.3
A solution for decoding is a row $\mathbf{u} = (\lambda, \psi)$ *of a matrix* $\mathbf{B}'$ *with* $\Phi(\mathbf{u})$ *having its leading position at the first element,* $\Phi(\mathbf{B}')$ *in weak Popov form and* $\mathbf{B}'$ *and*

$$\mathbf{B} = \begin{pmatrix} 1 & \hat{r} \\ 0 & M_\mathcal{G} \end{pmatrix}$$

being bases for the same module.

Proof. Leading position one implies fulfilment of the degree requirement described by Equation (3.11), because $\deg_q \Phi(\mathbf{B}')_{i,1} = \deg_q \mathbf{B}'_{i,1} + \mathbf{v}_1 \overset{LP=1}{\geq} \deg_q \mathbf{B}'_{i,j} + \mathbf{v}_j$, for the weights $\mathbf{v}$, within a row i.

To get a solution (λ, ψ) for Equation (3.10), it should hold:

$$\begin{aligned} & \lambda r = \psi + d \cdot M_{\mathcal{G}} \\ \Leftrightarrow \quad & \psi = \lambda \cdot r + c \cdot M_{\mathcal{G}} \qquad (\text{with } c = -d) \\ \Rightarrow \quad (\lambda,\ \psi) &= (a,\ a \cdot r + c \cdot M_{\mathcal{G}}) \quad (\text{with } \lambda = a) \\ &= a(1,\ r) + c(0,\ M_{\mathcal{G}}). \end{aligned}$$

The result is a submodule solution space with row basis $\mathbf{B}$. The theorem follows. □

Having a solution λ for Λ and ψ for $\Omega = \Lambda(f)$, only the retrieval step is left.

Retrieving the Information Polynomial

For Gao–like decoding of Gabidulin codes, as with Reed–Solomon codes, the final step of decoding is division. As q-polynomials are non–commutative, it is important to use left–division. If λ and ψ are valid solutions, it holds, that:

$$\lambda^{-1} \cdot \psi = \lambda^{-1} \cdot \Lambda(f) = \lambda^{-1} \cdot \Lambda \cdot f = f.$$

If the division fails, decoding failure is declared.

3.2.4 Interleaved Gabidulin Codes

As Reed–Solomon codes of Section 3.1.4, Gabidulin codes can be interleaved. Gabidulin codes can be interleaved horizontally or vertically. This section is an introduction to horizontally interleaved Gabidulin codes. This definition is the horizontally equivalent of [WZ13, Definition 2.17].

Definition 3.2.5 (Horizontally Interleaved Gabidulin Codes)
Let $\mathbf{y}_i = (g_1, g_2, \ldots, g_n)$ with $g_1, g_2, \ldots, g_n$ linear independent over $\mathbb{F}_{q^m}$, for every i with $1 \leq i \leq n$. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell$ over $\mathbb{F}_{q^m}$ be some ℓ, not necessarily distinct, Gabidulin codes. Let $\mathcal{C}_i$ have the code locators $\mathbf{y}_i$. The horizontally interleaved Gabidulin code $IGab[\ell; n, k_1, k_2, \ldots, k_\ell]$ is defined by:

$$IGab = \{(c_1, c_2, \ldots, c_\ell) = (f_1(\mathbf{y}_1), f_2(\mathbf{y}_2), \ldots, f_\ell(\mathbf{y}_\ell))\}.$$

Where $\deg_q f_i < k_i \leq n$ for all i, and $k_1, \ldots, k_\ell$ denounce the dimensions of the ℓ codes.

For $\ell = 1$, this results in a regular Gabidulin code.

Decoding of Interleaved Gabidulin Codes

Each codeword can of course be decoded on their own. But interleaving allows decoding in a slightly different way.

Definition 3.2.6 (Error Model of Interleaved Gabidulin Codes)
Let $\mathbf{c} = (c_1, c_2, \ldots, c_\ell)$ *be a codeword of some interleaved Gabidulin code* $\mathcal{C}_{IGab}$ *and let* $\mathbf{C} = \operatorname{ext} \mathbf{c}$. *The received word* $\mathbf{r}$ *is defined by*

$$\mathbf{r} = \operatorname{ext} \mathbf{R} = \mathbf{C} + \mathbf{E} = \operatorname{ext} \mathbf{c} + \operatorname{ext} \mathbf{e},$$

with some error matrix $\mathbf{E}$. *Let then be*

$$t = \operatorname{rank} (\mathbf{E}),$$

be an indicator of the error size.

For horizontally interleaved Gabidulin codes, the matrix representation $\mathbf{R}$ of the received word has dimensions $m \times \ell n$. This is important for the rank decomposition.

Theorem 3.2.4
There exists a vector $\mathbf{a} \in \mathcal{L}_q[x]^t$ *and a corresponding matrix* $\mathbf{B} \in \mathcal{L}_q[x]^{t \times \ell n}$ *fulfilling*

$$\mathbf{e} = \mathbf{aB}.$$

Proof. This is a straight forward implication of Lemma 3.2.1. □

This implies an error span polynomial $M_{\langle a \rangle}$ as in Definition 3.2.3 using the same characteristics. This error span polynomial is valid for all codewords.

Theorem 3.2.5
For every codeword of some interleaved Gabidulin code Theorem 3.2.2 creates a valid key equation using the same error span polynomial:

$$\Lambda(\hat{r}_i) \equiv \Lambda(f_i) \mod M_{\mathcal{G}_i}.$$

Proof. Theorem 3.2.4 implies a decomposition of every partial error $\mathbf{e}_i$, the error of the i-th codeword part c_i, of the form

$$\mathbf{e}_i = \mathbf{aB}_i.$$

Every codeword can be treated separately as a regular Gabidulin code codeword. This would create a key equation according to Theorem 3.2.2. As the error span polynomial only depends on $\mathbf{a}$ and every $\mathbf{a}$ is the same, all error span polynomials are identical. □

Theorem 3.2.5 creates a system of equations described by [LNPS15, Problem 1]. The solution space is described in [LNPS15, Lemma 1], which is the basis of the following theorem.

Theorem 3.2.6
The matrix $\mathbf{B}$ *is a basis for the multiple key equations of Theorem 3.2.5 without the degree requirements:*

$$\mathbf{B} = \begin{pmatrix} 1 & r_1 & r_2 & \dots & r_\ell \\ 0 & M_{\mathcal{G}_1} & 0 & \dots & 0 \\ 0 & 0 & M_{\mathcal{G}_2} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & M_{\mathcal{G}_\ell} \end{pmatrix}.$$

Proof. A vector $(\lambda, \psi_1, \psi_2, \dots, \psi_\ell) \in \mathbf{B}$ is of the form:

$$(a, ar_1 + c_1 M_{\mathcal{G}_1}, ar_2 + c_2 M_{\mathcal{G}_2}, \dots, ar_\ell + c_\ell M_{\mathcal{G}_\ell}).$$

This is an element–wise reformulation of the single key equations (cf. proof of Theorem 3.2.3). □

It is left to determine the degree requirements and model them in a useful way.

Theorem 3.2.7
The degree requirements of an interleaved Gabidulin code $IGab[\ell; n, k_1, \dots, k_\ell]$ *have the general form:*

$$\deg_q \lambda + \sum_{i=1}^{\ell} (k_i - 1) \geq \deg_q \psi_j + \sum_{i=1}^{\ell} (k_i - 1) - (k_j - 1).$$

Proof. The degree requirements of all codes $\mathcal{C}_1, \dots, \mathcal{C}_\ell$ can be listed in the following way:

$$\begin{aligned} \deg_q \lambda + (k_1 - 1) &\geq \deg_q \psi_1 \\ \deg_q \lambda + (k_2 - 1) &\geq \deg_q \psi_2 \\ \vdots \qquad &\qquad \vdots \\ \deg_q \lambda + (k_\ell - 1) &\geq \deg_q \psi_\ell. \end{aligned}$$

Adding to the i-th equation the sum of the deviations of all other equations

$$\sum_{j=1}^{i-1} (k_j - 1) + \sum_{j=i+1}^{\ell} (k_j - 1) = \sum_{j=1}^{\ell} (k_j - 1) - (k_i - 1),$$

results in the equation of the theorem. □

These degree requirements can be modelled either using a shift function Φ or a diagonal matrix $\mathbf{D}$ of the following form:

$$\Phi = (u_\lambda, u_1, \ldots, u_\ell) \longrightarrow (u_\lambda x^{[\gamma_\lambda]}, u_1 x^{[\gamma_1]}, \ldots, u_s x^{[\gamma_\ell]}),$$
$$\mathbf{D} = \operatorname{diag}(x^{[\gamma_\lambda]}, x^{[\gamma_1]}, \ldots, x^{[\gamma_\ell]}).$$

Where $\gamma_i = \sum_{j=0}^{\ell}(k_j - 1) - (k_i - 1)$ and $\gamma_\lambda = \sum_{j=0}^{\ell}(k_j - 1)$ are the shifts for the degree requirements.

Theorem 3.2.8
A vector $\mathbf{v} \in \mathcal{L}_q[x]^{s+1}$ is a solution for the key equations of Theorem 3.2.6 and degree requirements of Theorem 3.2.7, if $\mathbf{v}$ is in the submodule spanned by $\mathbf{B}$ and

$$LP(\Phi(\mathbf{v})) = 1.$$

Proof. $LP(\Phi(\mathbf{v})) = 1$ implies $\deg_q \Phi(\mathbf{v})_1 > \deg_q \Phi(\mathbf{v})_i$ with $i > 1$, as otherwise the leading position would be the largest i not fulfilling the equation. As $\deg_q \Phi(\mathbf{v}) = \deg_q \mathbf{v} + \deg_q \Phi((u_i = 1))_1$ modelled the degree requirements, the requirements are fulfilled.

As $\mathbf{B}$ is the solution space of the key equation, the theorem follows. □

The desired vector should be minimal in respect to its degree. To find a minimal vector module minimisation can be applied on the basis $\Phi(\mathbf{B}) = \mathbf{BD}$ to find a second basis $\mathbf{B}'$ containing the shortest vector.

If the solution is valid, the information polynomials can be retrieved by element–wise left–division by λ:

$$(\lambda, \psi_1, \psi_2, \ldots, \psi_\ell) \longrightarrow (\underbrace{\lambda^{-1}\psi_1}_{=f_1}, \underbrace{\lambda^{-1}\psi_2}_{=f_2}, \ldots, \underbrace{\lambda^{-1}\psi_\ell}_{=f_\ell})$$

If division fails, decoding failure is declared.

Minimization of the modules, to reach the final step, is the goal of the algorithms in Chapter 4 and Chapter 5.

An important remark on the presented strategy using horizontally interleaved Gabidulin codes is the unknown decoding radius. A shared decomposition might reduce decoding capabilities in terms of number of correctable errors.

3.2.5 Application of Interleaved Gabidulin Codes

One possible application of interleaved Gabidulin codes is random linear network coding (see [HMK+06]). This section will give a brief introduction for this topic.

Given an unknown network, a source and a sink, the goal of network coding is to transmit information from the source to the sink.

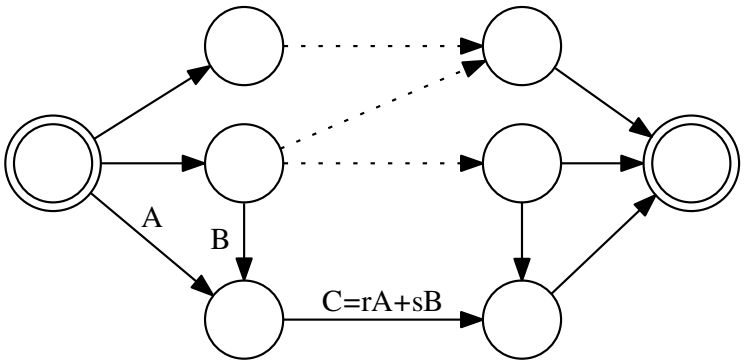

Figure 3.1: Example graph for random linear network coding. The dotted line represent an unknown network, r and s are random factors.

The source will send out rows of a codeword in matrix form, while the sink will collect incoming rows from nodes. A node will construct a new codeword by creating random linear combinations of incoming codewords using the formula:

$$\mathbf{c}_{new} = \sum_{i=1}^{s} r_i \mathbf{c}_i,$$

where r_i is a randomly chosen factor and c_i is the vector received on the i-th input line.

A widely used approach (cf. [KK08]) is constructing the original codeword with an identity matrix to record the particular linear combinations used:

$$\tilde{\mathbf{C}} = (\mathbf{I}\ \mathbf{C}).$$

In this context, interleaved Gabidulin codes might be able to reduce the overhead per transmission, by adding additional codewords:

$$\tilde{\mathbf{C}} = (\mathbf{I}\ \mathbf{C}_1\ \mathbf{C}_2\ \ldots\ \mathbf{C}_\ell).$$

A detailed examination of possible benefits and problems cause by this construction is yet to be done.

3.3 Summary

This chapter reviewed information on Gabidulin codes in context of their hamming metric equivalent, Reed–Solomon codes. For interleaved and non-interleaved codes, the general decoding procedure can be described in few essential steps:

1. Interpolation of the received word(s).
2. Solving the key equation.
3. Computation of the information polynomial.

This is essentially an informal description of [WZ13, Algorithm 3.6].

While steps 1 and 3 need specific attention, those steps have been described in [WZ13] and this chapter. [WZ13] approaches step 2 in its own way, but it can also be solved using module minimisation.

This allows construction of a more general framework spanning farther than Reed–Solomon and Gabidulin codes and embeds the decoding process for those codes in this framework. The algorithms of the following two chapters solve the problem of module minimisation over non–commutative polynomials, therefore solving the key equation for Gabidulin codes.

4 Known Algorithms

To apply the methods described in Section 3.1.3 and Section 3.2.3 algorithms to compute the weak Popov form of Section 2.4.2 are needed. In this section those algorithms will be explored for Ore extensions. **Section 4.1** will look at common factors and dependencies of the algorithms, like the complexity of multiplication. **Section 4.2** introduces the Ore extension variant of Mulders–Storjohann algorithm. Finally, **Section 4.3** presents the demand–driven variant of Mulders–Storjohann for decoding.

4.1 Preliminaries

The algorithms discussed partly use the same mechanics, the simple transformation, and they depend on the complexity of operations in the polynomial ring. These commonalities are examined first.

4.1.1 Complexity of Multiplication

A core operation of the algorithms in this chapter is multiplication of ring elements. While the number of additions is usually higher, the actual complexity of performing one multiplication usually outweighs the complexity of performing multiple additions. See [WZ13, Table 3.1] for q-polynomials.

In this case, polynomials are either regular polynomials $f \in \mathbb{F}[x]$, or polynomials from Ore extensions $g \in \mathbb{F}[x;\theta;\delta]$. Complexity of multiplication is different for those and dependent on the base field $\mathbb{F}$. Throughout this chapter, the complexity of multiplication, the asymptotic number of base field multiplications of two polynomials of degree maximal n shall be denoted by $M(n)$.

Following [CK91] regular polynomials of maximal degree n can be multiplied with complexity $M(n) = \mathcal{O}(n \log n \log\log n)$. If the base field supports fast Fourier transformation they can even be multiplied with complexity $M(n) = \mathcal{O}(n \log n)$.

[WZ13] gives multiplication algorithms and their complexity for q-polynomials. For two q-polynomials of degree m and n, [WZ13, Section 3.1.1] shows that the complexity of multiplication is $M(n,m) = \mathcal{O}((\max\{m,n\})^{1.69})$. For two q-polynomials modulo $x^{[n]} - x$, [WZ13, Section 3.1.3] gives an algorithm for multiplication with complexity $M(n) = \mathcal{O}(n^{1.69})$.

Complexity of Matrix Multiplication

Matrix multiplication is a central operation in Algorithm 4, its complexity is important. It is usually denoted by $\mathcal{O}(\ell^\omega)$ multiplications. A naive implementation uses $\omega = 3$, while there are known algorithms for $\omega \approx 2.374$ (cf. [CW90]), a usual implementation uses the Strassen algorithm of [Str69] with $\omega \approx 2.8$.

4.1.2 Simple Transformation

To achieve weak Popov form (cf. Definition 2.4.4) of a polynomial matrix, the leading positions of entries have to change. The main step to reach this goal is called simple transformation. The following Theorem 4.1.1, describing simple transformations for skew polynomials, is from [LNPS15, Definition 2], supplemented by a proof of its correctness.

Theorem 4.1.1
To eliminate the leading term of a row $\mathbf{v}$ *using another row* $\mathbf{u}$ *with* $\mathbf{v}, \mathbf{u} \in \mathbb{F}^n[x;\theta;\delta]\backslash\{0\}$ *and* $LP(\mathbf{v}) = LP(\mathbf{u})$ *a simple transformation of the form*

$$\begin{aligned}\mathbf{v} &= \mathbf{v} - \alpha x^\beta \mathbf{u}\\ \beta &= \deg \mathbf{v} - \deg \mathbf{u}\\ \alpha &= LC(LP(\mathbf{v}))/\theta^\beta(LC(LP(\mathbf{u})))\end{aligned}$$

can be applied. Where $LC(f)$ *is the leading coefficient and* $LP(v)$ *is the leading position.*

Proof. Let $\mathbf{u}_l, \mathbf{v}_l$ be the leading coefficient of the leading position of $\mathbf{u}, \mathbf{v}$. Then for

the leading term it holds, that:

$$\begin{aligned}
\mathbf{v}_l x^{\deg \mathbf{v}} - \alpha x^{\beta} \mathbf{u}_l x^{\deg \mathbf{u}} &\overset{Ore\odot}{=} \mathbf{v}_l x^{\deg \mathbf{v}} - \alpha(\theta(\mathbf{u}_l) x^{\beta-1} \mathbf{u}_l x + \delta(\mathbf{u}_l)) x^{\deg \mathbf{u}} \\
&\dots \\
&= \mathbf{v}_l x^{\deg \mathbf{v}} - \alpha \theta^{\beta}(\mathbf{u}_l) x^{\beta + \deg \mathbf{u}} && + \mathcal{O}(x^{\beta + \deg \mathbf{u} - 1}) \\
&= \mathbf{v}_l x^{\deg \mathbf{v}} - \alpha \theta^{\beta}(\mathbf{u}_l) x^{\deg \mathbf{v} - \deg \mathbf{u} + \deg \mathbf{u}} && + \mathcal{O}(x^{\beta + \deg \mathbf{u} - 1}) \\
&= \mathbf{v}_l x^{\deg \mathbf{v}} - \alpha \theta^{\beta}(\mathbf{u}_l) x^{\deg \mathbf{v}} && + \mathcal{O}(x^{\deg \mathbf{v} - 1}) \\
&= (\mathbf{v}_l - \alpha \theta^{\beta}(\mathbf{u}_l)) x^{\deg \mathbf{v}} && + \mathcal{O}(x^{\deg \mathbf{v} - 1}).
\end{aligned}$$

Where $\mathcal{O}(x^{\beta + \deg \mathbf{u} - 1})$ is the term introduced by the derivative δ with degree strictly smaller than $\deg \mathbf{v}$. For the leading term to be eliminated, only the coefficient of $x^{\deg v}$ has to be zero. This is equivalent to:

$$\begin{aligned}
& 0 = (\mathbf{v}_l - \alpha \theta^{\beta}(\mathbf{u}_l)) x^{\deg \mathbf{u}} \\
\Rightarrow \quad & 0 = \mathbf{v}_l - \alpha \theta^{\beta}(\mathbf{u}_l) \\
\Leftrightarrow \quad & \alpha = \mathbf{v}_l / \theta^{\beta}(\mathbf{u}_l)).
\end{aligned}$$

In conclusion, the leading term is eliminated. □

The behaviour of simple transformations can further be specified by the following lemma.

Lemma 4.1.1
A simple transformation will either:

1. *Lower the degree of the row, or*
2. *Lower the leading position of the row.*

Proof. Let $\mathbf{u}, \mathbf{v} \in \mathbb{F}[x; \theta; \delta]^{\ell}$ be two rows used in a simple transformation with $\deg \mathbf{u} \geq \deg \mathbf{u}$, so that $\mathbf{u} = \mathbf{u} - \alpha x^{\beta} \mathbf{v}$ is a valid simple transformation. The leading positions of the two rows are identical and shall be denoted by $j = LP(\mathbf{u}) = LP(\mathbf{v})$.

A simple transformation can not introduce a new term of degree $\deg \mathbf{u}$ at the right of the current leading position, because $\beta + \deg \mathbf{v}_{j+\eta} < \deg \mathbf{U}_j$ for all $\eta \in \mathbb{N}$. If this was not the case, the leading position of $\mathbf{v}$ had to be $LP(\mathbf{v}) = j + \eta$. A term of the same degree can therefore only exist on lower positions.

As a simple transformation eliminates the leading term of the leading position there can either be another term of the same degree to the left, or there is none. If there is no term of the same degree as the former leading position, the degree of the row drops by at least one. If there is a term of the same degree, the leading position will shift to this term. □

A lowering of the leading position is informally a move to the left for the leading position. This is an important observation for correctness and complexity proofs of the algorithms.

Complexity of Simple Transformations

A simple transformation is worst case $\mathcal{O}(lM(n))$ as l elements get multiplied and subtracted. But for special cases it can be much better. For this estimation the length of a polynomial is useful.

Definition 4.1.1 (Length of a Polynomial)
Let $f \in \mathbb{F}[x;\theta;\delta]$ be some polynomial with the set of coefficients $\{f_0, f_1, f_2, \ldots, f_{\deg f}\}$. The length $L(f)$ of the polynomial is given by

$$L(f) = |\{i : f_i \neq 0\}|.$$

Informally $L(f)$ is the number of terms of a polynomial f with coefficient not zero.

Lemma 4.1.2
Let $f, g \in \mathbb{F}[x;\theta;0]$ be a polynomials of some Ore extension with derivative zero and $\deg f \geq \deg g$. Let further

$$\alpha = \frac{f_{\deg f}}{\theta^{\beta}(g_{\deg g})}, \beta = \deg f - \deg g$$

be constants depending on f and g, where f_i is the coefficient of the monomial of degree i. The composition $f - \alpha x^{\beta} g$ can then be computed in $\mathcal{O}(L(g))$.

Proof. The multiplication $\alpha x^{\beta} g$ is carried out for every term of g and results in one computation of $\theta^{\beta}(g_i)$ and one multiplication with α for every coefficient. Coefficients that equal zero can be ignored.

Addition, and therefore subtraction, is carried out element–wise, only non–zero coefficients need to be considered. As g has exactly $L(g)$ non–zero coefficients, the theorem follows. □

Restrictions for Decoding

The matrix for decoding Gao–like has a unique form, so complexity can be estimated more accurately for this special case. One restriction is the Ore extension itself. Complexity analysis will be done for $\mathbb{F}[x;\theta;\delta]$ with $\theta(a) = a^q$ and $\delta = 0$.

Definition 4.1.2 (Form of Decoding Matrix)
Let $\mathbf{B} \in \mathbb{F}[x;\theta;\delta]^{\ell\times\ell}$ *with* $\theta(a) = a^q$ *and* $\delta = 0$ *be a basis in matrix representation and* $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_\ell) \in \mathbb{N}_0^\ell$ *a vector of weights. The matrix of interest has the form:*

$$\mathbf{B} = \begin{pmatrix} 1x^{\mathbf{v}_1} & s_1x^{\mathbf{v}_2} & s_2x^{\mathbf{v}_3} & \ldots & s_{\ell-1}x^{\mathbf{v}_\ell} \\ 0 & G_1x^{\mathbf{v}_2} & 0 & \ldots & 0 \\ 0 & 0 & G_2x^{\mathbf{v}_3} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & G_{\ell-1}x^{\mathbf{v}_\ell} \end{pmatrix}.$$

It is assumed that $\deg s_i \leq \deg G_i$ as calculating using $s_i \mod G_i$ holds the same results.

This matrix has multiple useful properties that simplify computation over those. Besides the first row, the leading positions are on the diagonal. One important conclusion from this is, that there is only one conflict and that the matrix is in weak Popov form, as soon as one rows leading position moves to the first column. Lemma 4.1.3 adds another property that will be used, as the given matrix fulfils the condition of only having one leading position conflict.

Lemma 4.1.3
Let $\mathbf{B} \in \mathbb{F}[x;\theta;\delta]^{m\times\ell}$ *be some matrix with only two rows having the same leading position, excluding zero rows. Applying simple transformations will not increase the number of leading position conflicts.*

Proof. As there is only one conflict, both rows will be used during a simple transformation. If the conflict is not resolved, both leading positions will remain. If the conflict is resolved, only one leading position will change. As every other leading position is distinct, the new value can conflict with at most one. □

These properties will be used for complexity analysis, as they allow more precise estimations on the number of simple transformations done within the algorithms.

Finally, a useful lemma for a matrix of the given form is the following:

Lemma 4.1.4
For a matrix $\mathbf{B} \in \mathbb{F}[x;a^q;0]^{\ell\times\ell}$ *and a vector* $\mathbf{v} \in \mathbb{N}_0^\ell$ *as described in Definition 4.1.2 it holds, that:*

$$\Delta\mathbf{B} \leq \deg \mathbf{B}_1 - \mathbf{v}_1.$$

Proof. As $\mathbf{B}$ is in upper triangular form, the degree of its determinant is the sum of

the degrees of the diagonal elements:

$$\begin{aligned}\deg\det\mathbf{B} &= \sum_{i=1}^{\ell}\deg\mathbf{B}_{i,i}\\ &= \sum_{i=1}^{\ell-1}\deg(G_i+\mathbf{v}_{i+1})+\deg\mathbf{v}_1.\end{aligned}$$

The first element of the matrix can be ignored, as it has degree zero besides its weight. The orthogonality defect can therefore be rewritten to

$$\begin{aligned}\Delta\mathbf{B} &= \deg\mathbf{B}-\deg\det\mathbf{B}\\ &= \deg\mathbf{B}-(\sum_{i=1}^{\ell-1}(\deg G_i+\mathbf{v}_{i+1})+\mathbf{v}_1)\\ &= \sum_{i=1}^{\ell}\deg\mathbf{B}_i-\sum_{i=1}^{\ell-1}(\deg G_i+\mathbf{v}_{i+1})-\mathbf{v}_1\\ &= \deg\mathbf{B}_1+\sum_{i=2}^{\ell}(\deg G_{i-1}+\mathbf{v}_i)-\sum_{i=1}^{\ell-1}(\deg G_i+\mathbf{v}_{i+1})-\mathbf{v}_1\\ &= \deg\mathbf{B}_1-\mathbf{v}_1.\end{aligned}$$

This is the claim of the lemma. □

To comply with the notation used in [LNPS15] the following theorem introduces the variable μ and an important relationship to the orthogonality defect.

Theorem 4.1.2
Let $\mu = \max_i\{\deg\mathbf{B}_{i,i}-\mathbf{v}_i\}$ for the matrix of Definition 4.1.2 with the weights $\mathbf{v}$. It then holds, that:

$$\Delta\mathbf{B}\leq\mu-\mathbf{v}_1.$$

Proof. It holds, that $\max_i\deg\mathbf{B}_{i,i}\geq\deg\mathbf{B}_1$ as $\deg s_i\leq\deg G_i$ was assumed. Together with Lemma 4.1.4 the theorem follows. □

4.2 Mulders–Storjohann Algorithm

The Mulders–Storjohann algorithm was introduced by [MS03] for regular polynomials $f\in\mathbb{F}[x]$ over some field $\mathbb{F}$. [LNPS15] modified the algorithm for skew polynomials $g\in\mathbb{F}[x;\theta;\delta]$ of some Ore extension.

4.2.1 Algorithm

Algorithm 1 has a simple structure: As long as the matrix is not in weak Popov form, it applies a simple transformation (Line 4 to Line 7), using two rows with the same leading position (Line 3). LC refers to the leading coefficient and LP to the leading position introduced in Definition 2.4.3.

Algorithm 1: MS(B), Mulders–Storjohann Algorithm.

Input: Basis $\mathbf{B} \in \mathbb{F}[x]^{m \times \ell}$
Output: Matrix $R(\mathbf{B}) \in \mathbb{F}[x]^{m \times \ell}$ so that $R(\mathbf{B})$ is in weak Popov form.

1 let U = B
2 **while** U *is* ***not*** *in weak Popov form* **do**
3 find i_1, i_2 so that $LP(\mathbf{U}_{i_1}) = LP(\mathbf{U}_{i_2})$ and $\deg \mathbf{U}_{i_1} \geq \deg \mathbf{U}_{i_2}$
4 let $j = LP(\mathbf{U}_{i_1})$
5 let $\beta = \deg \mathbf{U}_{i_1} - \deg \mathbf{U}_{i_2}$
6 let $\alpha = LC(\mathbf{U}_{i_1,j}) / \theta^{\beta}(LC(\mathbf{U}_{i_2,j}))$
7 let $\mathbf{U}_{i_1} = \mathbf{U}_{i_1} - \alpha x^{\beta} \mathbf{U}_{i_2}$
8 **return** U

The termination and correctness of Algorithm 1 is analogue to [LNPS15, Theorem 1].

Theorem 4.2.1
Algorithm 1 terminates.

Proof. As long as the matrix is not in weak Popov form, there are two or more rows with the same leading position, so it is possible to find two conflicting rows in Line 3. The condition $\deg \mathbf{U}_{i_1} \geq \deg \mathbf{U}_{i_2}$ is no loss of generality, as i_1 and i_2 can be switched.

Consider the following weighting functions for some vector $\mathbf{v} \in \mathbb{F}[x]^{\ell}$ and some matrix $\mathbf{M} \in \mathbb{F}[x]^{m \times \ell}$:

$$\varphi(\mathbf{v}) = \ell \deg \mathbf{v} + LP(\mathbf{v}) \qquad \text{with } \varphi(0) = 0,$$
$$\varphi_M(\mathbf{M}) = \sum_{i=1}^{m} \varphi(\mathbf{M}_i).$$

The function φ, and therefore the sum φ_M, is lower bounded by zero. At the beginning of Algorithm 1, $\varphi_M(\mathbf{M})$ starts with a finite value. Lemma 4.1.1 states, that a simple transformation either lowers the leading position of $\mathbf{v}$, if there is another position with equal degree, or it lowers the degree of $\mathbf{v}$. If the degree is reduced, the leading position might increase. In the worst case it might increase by

$\ell - 1$, from position one to the maximum position. So φ decreases by at least one for every simple transformation including $\mathbf{v}$, as a decrease in degree would decrease the value of φ by ℓ.

As for every row $\mathbf{v}$ the starting value is finite, the sum of those values φ_M is finite as well. Every simple transformation changes some row $\mathbf{M}_i$ of $\mathbf{M}$, as $\varphi(\mathbf{M}_i)$ decreases by at least one, so does φ_M. Since φ_M is finite at the start, is lower bounded by zero, and reduced by at least one in every step, there can only be finitely many steps. □

Using Theorem 4.2.1 the correctness of Algorithm 1 is proven easily.

Theorem 4.2.2
Algorithm 1 is correct.

Proof. Algorithm 1 is correct, as it only terminates if and only if the matrix is in weak Popov form, otherwise the loop would continue. In combination with Theorem 4.2.1, the algorithm is correct. □

4.2.2 Complexity

[LNPS15] shows the following theorem on the complexity of Algorithm 1.

Theorem 4.2.3
Algorithm 1 can compute the weak Popov form for decoding of Gabidulin codes in $\mathcal{O}(\ell^2\mu^2)$ *with* $\mu = \max_i\{\deg \mathbf{B}_{i,i} - \mathbf{v}_i\}$.

Proof. One simple transformation costs $\mathcal{O}(\ell)$ multiplications by the scaling factor αx^β and subtractions in the base ring as well as one computation of θ^β. Finding some conflicting pair of rows and their leading position can be done in $\mathcal{O}(\ell)$ with a one time preparation of $\mathcal{O}(\ell^2)$.

The algorithm does at most $\mathcal{O}(\ell(\Delta\mathbf{M} + 1))$ many simple transformations, as for the

starting matrix $\mathbf{M}$ and the resulting matrix $\mathbf{V}$ it holds, that:

$$\begin{aligned}
\varphi_M(\mathbf{M}) - \varphi_M(\mathbf{V}) &= \sum_{i=1}^{\ell} \varphi(\mathbf{m_i}) - \varphi(\mathbf{v_i}) \\
&= \sum_{i=1}^{\ell} \ell \deg \mathbf{m_i} + LP(\mathbf{m_i}) - (\ell \deg \mathbf{v_i} + LP(\mathbf{v_i})) \\
&= \ell(\sum_{i=1}^{\ell} \deg \mathbf{m_i} - \deg \mathbf{v_i}) + \underbrace{\sum_{i=1}^{\ell} LP(\mathbf{m_i}) - LP(\mathbf{v_i})}_{(1)} \\
&= \ell(\sum_{i=1}^{\ell} \deg \mathbf{m_i} - \deg \mathbf{v_i}) + \underbrace{LP(\mathbf{m_0}) - 1}_{<\ell} \\
&< \ell(\deg \mathbf{M} - \deg \mathbf{V} + 1) \\
&\overset{(2)}{=} \ell(\Delta(\mathbf{M}) + 1).
\end{aligned}$$

The term at (1) is equal to $LP(\mathbf{m_0}) - 1$, because the form of the input matrix has only diagonal elements outside of row one, so their leading position must be on the diagonal. As $\mathbf{V}$ is in weak Popov form, all leading positions are different. So every leading position of two to ℓ cancels out and only the leading positions of row one in $\mathbf{M}$ and of value one in $\mathbf{V}$ are left.

Step (2) is valid, as $\deg \mathbf{V} = \deg \det \mathbf{V} = \deg \det \mathbf{M}$, because simple transformations do not change the degree of the determinant.

Theorem 4.1.2 showed that the orthogonality defect can be approximated by μ. The complexities of a simple transformation and the count of simple transformations can now be multiplied together. This results in $\mathcal{O}(\ell^2\mu^2)$. □

This complexity can be improved by a variant specialised for decoding: The demand–driven algorithm.

4.3 Demand–Driven Algorithm

[Nie13a] presented a variant of the Mulders–Storjohann algorithm over regular polynomials $\mathbb{F}[x]$, called the demand–driven algorithm. This algorithm is converted to Ore extensions $\mathbb{F}[x;\theta;\delta]$ in [LNPS15, Section 5]. While in the general case it has a complexity of $\mathcal{O}(\ell\mu^3)$, [LNPS15, Theorem 3] improves the complexity for decoding of Gabidulin codes to $\mathcal{O}(\ell\mu^2)$ when used with $M_{\mathcal{G}_i} = x^{[m]} - x$.

4.3.1 Algorithm

The algorithm as in [LNPS15, Section 5] is given by Algorithm 2 with slight modifications for names and the return value, as the single solution λ_1 is sufficient to return.

Algorithm 2: DD($\tilde{\mathbf{s}}, \tilde{\mathbf{G}}, \mathbf{v}$), Demand–Driven Algorithm.

Input: $\tilde{s}_i = s_i x^{v_{i+1}}$ and $\tilde{G}_i = G_i x^{v_i}$ for $i = 1, \ldots, \ell - 1$
Output: λ solution candidate for Λ

```
1  let (d,p) = (deg, LP) of (x^{v_1}, s̃_1, ..., s̃_{ℓ-1})
2  if p = 1 then
3      return 1
4  let (λ_1, ..., λ_ℓ) = (x^{v_1}, 0, ..., 0)
5  let α_i x^{d_i} = leading monomial of G̃_i for i = 1, ..., ℓ - 1
6  while deg_q λ_1 ≤ d do
7      let α = coefficient of x^d in (λ_1 s̃_p mod G̃_p)
8      if α ≠ 0 then
9          if d < d_p then
10             swap((λ_1, α, d),(λ_p, α_p, d_p))
11         let β = d - d_p
12         let λ_1 = λ_1 - α/(θ^β(α_p)) x^β λ_p
13     let (d, p) = (d, p - 1)
14     if p = 1 then
15         let (d, p) = (d - 1, ℓ)
16 return λ_1 x^{-v_1}
```

As Algorithm 2 only returns a solution candidate λ for Λ it is important to clarify how to calculate the information polynomials.

Theorem 4.3.1
It holds, that:

$$f_i = \lambda^{-1}(\lambda r_i \mod M_{\mathcal{G}_i}).$$

Proof. The key equation Equation (3.10) states, that $\Lambda(\hat{r}(x)) \equiv \Lambda(f(x)) \mod M_{\mathcal{G}}$. If λ is a solution for Λ the theorem follows. □

This relation is also important throughout the algorithm.

Functionality

The algorithm uses the special purpose it is designed to solve to ignore data points that are not needed to be calculated. The first column of the matrix that is to be transformed into weak Popov form, will contain the solution λ, so only the first column will be stored. Additional values are computed on demand (Line 7) as elements within the vector have the given form (cf. Theorem 3.2.6).

One of the main ideas is to restrict calculation to the first row of the matrix. In the beginning, all other rows have leading positions on the diagonal, so only the first row has to be processed. Whenever a simple transformation is computed (Line 12), line one should be involved. For this to happen, the target row is switched so it is row one (Line 10). This will keep the leading positions on all rows but the first on the diagonal.

The algorithm finds the new leading position by stepping through all possible valued of the φ function of Theorem 4.2.1, by either reducing the degree by one and setting the possible leading position to the end of the row $(d-1, \ell)$, when the leading position would have reached column one, or reducing the possible leading position $(d, p-1)$. If the leading position actually reached column one, the degree of the solution would need to be bigger than the last degree and is caught by the while condition.

Theorem 4.3.2
Algorithm 2 is correct, as it computes the minimal solution λ to the decoding problem of Section 4.1.2.

Proof. Equivalent to Theorem 4.3.1, the computation $\lambda_1 \tilde{s}_p \mod \tilde{G}_p$ results in a valid solution of the key equation without the degree requirement.

The algorithm stops, the first time the solution λ_1 has the highest degree in its row. This implies leading position one and the termination criterion. As this would result in a matrix in weak Popov form, the element has to be minimal.

As a variant of Mulders-Storjohann, the algorithm terminates with the same argument as the Mulders-Storjohann algorithm: It steps backwards through the helping function φ_M, which is finite in the beginning. □

As the algorithm results in a correct solution candidate, its complexity is of importance, to be comparable to the Mulders–Storjohann algorithm.

4.3.2 Complexity

[LNPS15] uses a helper function to describing the complexity of the algorithm to distinguish between two cases resulting in different complexities.

Definition 4.3.1 (Helper Function ρ)
Let L is the length of a polynomial of Definition 4.1.1 and LT the leading term of a polynomial. The helper function ρ is defined by:

$$\rho = \begin{cases} \max\{L(G_i)\} & if\ \forall i : LT(G_i) = G_i|_{\frac{\deg G_i}{2}}, \\ \mu & otherwise. \end{cases} .$$

Informally, the condition describes that the degree of the second highest term of G_i has to be lower than half the degree of G_i.

The lemma making the differentiation useful is the following taken from [LNPS15, Theorem 3].

Lemma 4.3.1
Let $a, g \in \mathbb{F}[x; \theta; \delta]$ be non-commutative polynomials with Ore derivative $\delta = 0$, where a, g fulfil the conditions:

$$LT(g) = g|_{\frac{\deg g}{2}}$$
$$\deg a < k + \deg g.$$

Then at most $L(g)+1$ coefficients need to be computed to compute the k-th coefficient $b_k = (a \mod g)_k$.

Proof. The computation is based around the degree k, and obviously a_k is a necessary factor.

Consider the set $S := \{j : g_j \neq 0, j \neq \deg g\}$ of indices of non–zero coefficients of the modulus g. Then all degrees of the form $k + \deg g - i$ with $i \in S$, are part of the computation for b_k.

In general, all relevant factors have the form $k + t \deg g - i_1 - \ldots - i_t < \deg a$ with $t \in \mathbb{N}_0, i_\ell \in S$. The degree requirement $\deg a < k + \deg g$ creates a contradiction for

$t \geq 2$ as:

$$\begin{aligned}\deg a &> k + 2\deg g - i_1 - i_2 \\ &> k + 2\deg g - 2 \cdot \frac{\deg g}{2} \\ &= k + \deg g \\ &> \deg a.\end{aligned}$$

It follows, that $t = 1$. Together with $|S| \leq L(g)$, this implies the lemma. □

The theorem on the complexity of the demand–driven algorithm is given by [LNPS15, Theorem 3] and is restated here.

Theorem 4.3.3
Algorithm 2 has a complexity of $\mathcal{O}(\ell\mu^2\rho)$ over some Ore extension with derivative zero.

Proof. Besides Line 12 and Line 7, every other line is essentially $\mathcal{O}(1)$ (reducing by one, fixed computations of length one, swapping two tuples of length three). Or they can be safely ignored, as they are $\mathcal{O}(\ell)$ and not contained in the loop: Initialising a tuple of length ℓ and extracting ℓ leading monomials.

Line 12 is a simple transformation on a single element, its complexity is $\mathcal{O}(\mu)$ by Lemma 4.1.2. As the algorithm acts similar to the Mulders-Storjohann algorithm, its main loop will iterate at most $\mathcal{O}(\ell\mu)$ times (cf. Theorem 4.2.3).

Line 7 has a complexity of $\mathcal{O}(\mu\rho)$. If the condition for $\rho \neq \mu$ is not fulfilled, the complete polynomial is computed in $\mathcal{O}(\mu^2)$. Otherwise Lemma 4.3.1 states, as $\deg \lambda_1 s_i < 2 \cdot \deg g_i$, that at most $L(\tilde{G}_i) + 1$ coefficients of $\lambda_1 \tilde{s}_i$ make up the computation of the coefficient for x^d. As a single one can be computed in $\mathcal{O}(\deg g)$=$\mathcal{O}(\mu)$, they can therefore be computed entirely in $\mathcal{O}(\mu L(\tilde{G}_i))$ operations over $\mathbb{F}_{q^m}$.

This results in a complexity of $\mathcal{O}(\ell\mu^2\rho)$ as claimed by the theorem. □

This results in a complexity of $\mathcal{O}(\ell\mu^2)$ for Gao–like decoding using $G_i = x^{[m]} - x$ and a complexity of $\mathcal{O}(\ell\mu^3)$, using unknown polynomials G_i.

4.4 Summary

This chapter described two algorithms already converted to non–commutative polynomials and their complexity analysis. The Mulders-Storjohann algorithm gives a fairly simple approach to computing the weak Popov form in $\mathcal{O}(\ell^2 n^2)$ and the demand–driven algorithm builds on those results for the special case of decoding, only.

This results in a n-ℓ trade–off for general decoding and in a plain improvement for a special decoding case. This improvement uses a modulus where the second term has a degree lower than half the degree of the modulus, for example $x^{[m]} - x$ often used for Gao–like decoding. The resulting complexity was $\mathcal{O}(\ell n^2)$ in the special case. The next chapter examines an ℓ-n trade–off by converting Alekhnovich's algorithm.

5 Alekhnovich's Algorithm

In 2002, Alekhnovich introduced an algorithm to compute a reduced basis over regular polynomial rings in [Ale02]. The paper was refined and also released as [Ale05]. In this chapter, the algorithm of Alekhnovich will be introduced for non–commutative polynomials analogue to the structure of its original introduction: In Section 5.1 the basic operator and a correct algorithm will be described and examined. In the following sections an approximation will be examined for non–commutative polynomials and applied to the algorithm, allowing a divide and conquer strategy to improve the algorithms runtime.

5.1 Basic Algorithm

Alekhnovich's algorithm can be understood as a variant of the Mulders–Storjohann algorithm, it utilises the same concept of simple transformations. A major difference is its output. Alekhnovich's algorithm produces a matrix $\mathbf{U}$ such that the input matrix $\mathbf{B}$ can be transformed to weak Popov form by multiplying: $\mathbf{UB}$.

Definition 5.1.1 (Singular Element Matrix)
A matrix $\mathbf{E}[a,b]$ *will be called singular element matrix, if it is of the form*

$$\mathbf{E}[a,b] = (e_{i,j} = \left\{ \begin{array}{ll} 1 & if\ (i,j) = (a,b) \\ 0 & otherwise \end{array} \right).$$

Informally, it is a matrix with only one entry and zeros everywhere else.

Alekhnovich's algorithm makes heavy use of matrix multiplication, which can be modelled differently in practice, but their functionality is important, as they accumulate simple transformations.

Lemma 5.1.1
Let $\mathbf{T}, \mathbf{I}, \mathbf{E} \in \mathbb{F}[x;\theta;\delta]^{m\times m}, \mathbf{B} \in \mathbb{F}[x;\theta;\delta]^{m\times \ell}$ *be some matrices with* $\mathbf{E}[a,b]$ *a singular element matrix and* $\mathbf{I}$ *the identity matrix. Multiplying* $\mathbf{T} = \mathbf{I} + \alpha\mathbf{E}[a,b]$ *on the left to* $\mathbf{B}$*,* $\mathbf{TB}$*, will compute an elementary row transformation* $\mathbf{B}_a = \mathbf{B}_a + \alpha\mathbf{B}_b$*.*

Proof. Let $\mathbf{B} \in \mathbb{F}[x;\theta;\delta]^{m\times\ell}$ be some matrix to compute a row transformation on:

$$\begin{aligned}\mathbf{T}\cdot\mathbf{B} &= (c_{i,j} = \sum_{k=1}^{m} \underbrace{\mathbf{T}_{i,k}}_{=0 \text{ for } i\neq k \text{ and } (i,k)\neq(a,b)} \cdot\mathbf{B}_{k,j}) \\ &= (c_{i,j} = \begin{cases} \mathbf{B}_{i,j} & \text{if } i \neq a \\ \mathbf{B}_{a,j} + \mathbf{T}_{a,b}\mathbf{B}_{b,j} & \text{else} \end{cases}).\end{aligned}$$

An entry in $\mathbf{T}_{a,b}$ will therefore compute an elementary transformation on row a using row b: $\mathbf{B}_a = \mathbf{B}_a + \mathbf{T}_{a,b}\mathbf{B}_b$. □

The actual algorithm is split in two parts: An elementary reduction operator, Algorithm 3, and an inductive reduction operator, Algorithm 4. The goal of the elementary reduction operator is to create a matrix that, if left multiplied on the input matrix, will reduce the degree of the input matrix by one or transform it into weak Popov form.

Algorithm 3: $R(B)$, the elementary reduction operator.

Input: Basis $\mathbf{B} \in \mathbb{F}[x]^{m\times\ell}$
Output: Matrix $R(\mathbf{B}) \in \mathbb{F}[x]^{m\times m}$ so that either $R(\mathbf{B})\cdot\mathbf{B}$ is in weak Popov form or $\deg(R(\mathbf{B})\cdot\mathbf{B}) < \deg\mathbf{B}$.

```
1  let n = deg B
2  let U = I
3  while deg B = n do
4      if B is in weak Popov form then
5          return U
6      find i_1, i_2 so that LP(B_{i_1}) = LP(B_{i_2}) and deg B_{i_1} ≥ deg B_{i_2}
7      let j = LP(B_{i_1})
8      let β = deg B_{i_1} − deg B_{i_2}
9      let α = LC(B_{i_1,j}) / θ^β(LC(B_{i_2,j}))
10     let T = I − α x^β E_{i_1 i_2}
11     let U = T · U
12     let B = T · B
13 return U
```

The goal of the inductive reduction operator is to make use of the elementary operator to create a matrix that, if left multiplied to the input matrix, would reduce the degree of the input matrix by t or transform it into weak Popov form.

Algorithm 4: $R(B, t)$, the inductive reduction operator.

Input: Basis $\mathbf{B} \in \mathbb{F}[x]^{m \times \ell}$
Output: Matrix $R(\mathbf{B}) \in \mathbb{F}[x]^{m \times m}$ so that either $R(\mathbf{B}) \cdot \mathbf{B}$ is in weak Popov form or $\deg(R(\mathbf{B}) \cdot \mathbf{B}) < \deg \mathbf{B}$.

```
1 let n = deg B
2 let U = I
3 while deg U · B > n − t do
4 |   if U · B is in weak Popov form then
5 |   |   return U
6 |   else
7 |   |   let U = R(U · B) · U
8 return U
```

It is easy to see, that Algorithm 4, if it is correct, with input matrix $\mathbf{B}$ and $t = \deg \mathbf{B}$ will result in a matrix that can be left multiplied on $\mathbf{B}$ to bring it into weak Popov form.

5.1.1 Correctness

Using Lemma 5.1.1, Algorithm 3 can be evaluated for correctness.

Theorem 5.1.1
Let $\mathbf{B} \in \mathbb{F}[x; \theta; \delta]^{m \times \ell}$ *be some matrix over some Ore extension, then Algorithm 3* $R(B)$ *computes a matrix* $\mathbf{U} \in \mathbb{F}[x; \theta; \delta]^{m \times m}$ *such that* $\mathbf{UB}$ *has* $\deg \mathbf{UB} < \deg \mathbf{B}$ *or* $\mathbf{UB}$ *is in weak Popov form.*

Proof. If $\mathbf{B}$ is in weak Popov form in the beginning, the algorithm will terminate immediately. So we assume that $\mathbf{B}$ is not in weak Popov form in the beginning. As long as the matrix $\mathbf{B}$ is not in weak Popov form, a simple transformation can be computed.

Lemma 5.1.1 states, that a matrix multiplication can be equivalent to applying an elementary row transformation. Line 10 creates a matrix of the necessary form of the multiplication matrix of Lemma 5.1.1, with the multiplication constant of a simple transformation in Theorem 4.1.1. Line 11 is therefore an application of a simple transformation on $\mathbf{U}$ and Line 12 on $\mathbf{B}$.

Matrix multiplication is associative as long as the contained elements are associative. As Ore extensions are associative [Ore33, Section 1], it follows, for the original

matrix $\mathbf{B}$ and the application of n simple transformations $\mathbf{T}_1, \ldots, \mathbf{T}_n$, where $\mathbf{U}_i = \mathbf{T}_i \cdot \mathbf{T}_{i-1} \cdot \ldots \cdot \mathbf{T}_1$:

$$\begin{aligned}\mathbf{U}_n \cdot \mathbf{B} &= \mathbf{T}_n \cdot \mathbf{U}_{n-1} \cdot \mathbf{B} \\ &= \mathbf{T}_n \cdot \mathbf{T}_{n-1} \cdot \ldots \cdot \mathbf{T}_1 \mathbf{B} \\ &= (\mathbf{T}_n \cdot (\mathbf{T}_{n-1} \cdot (\ldots \cdot (\mathbf{T}_1 \mathbf{B}) \ldots))).\end{aligned}$$

Multiplication by $\mathbf{U}_n$ is therefore equivalent to sequential application of n simple transformations.

As established in Lemma 4.1.1, a simple transformation will either reduce the degree of the target row, therefore reducing the degree of the matrix, or it will reduce the leading position. If the degree is decreased, the algorithm terminates. If the leading position is reduced, the matrix might have turned into weak Popov form. If the matrix is not in weak Popov form, the conditions for further simple transformations are kept, due to the only condition being the existence of a leading position conflict. After at most $m\ell$ simple transformations the degree has to decrease. □

Based on the correctness of its building block Algorithm 3, Algorithm 4 can be evaluated for correctness.

Theorem 5.1.2
Let $\mathbf{B} \in \mathbb{F}[x;\theta;\delta]^{m\times\ell}$ be some matrix over some Ore extension, then Algorithm 4 $R(\mathbf{B}, t)$ computes a matrix $\mathbf{U} \in \mathbb{F}[x;\theta;\delta]^{m\times m}$ such that $\mathbf{UB}$ has $\deg \mathbf{UB} \leq \deg \mathbf{B} - t$ or $\mathbf{UB}$ is in weak Popov form.

Proof. Algorithm 3 gives a matrix $\mathbf{U}$ that reduces the degree of $\mathbf{B}$ by at least one or brings it into weak Popov form. As the degree can no longer be reduced, if the matrix is in weak Popov form, the algorithm will terminate if $\mathbf{UB}$ is in weak Popov form.

Due to the non commutativity the multiplication order is important. As $R(\mathbf{UB})$ will return a matrix that is left multiplied on $\mathbf{UB}$, it has to be left multiplied on $\mathbf{U}$. The resulting chain will have the form $\mathbf{U_t U_{t-1}} \ldots \mathbf{U_2 U_1 B}$. After at most t calls to Algorithm 3, the matrix $\mathbf{UB}$ will be in weak Popov form or it will hold that $\deg \mathbf{UB} \leq \deg \mathbf{B} - t$, as every call reduces the degree by at least one. □

Theorem 5.1.3
Algorithm 4 terminates.

Proof. Algorithm 3 terminates as there are at most $m\ell$ steps to compute until the degree has to decrease. As the degree is finite and lower bounded by zero, Algo-

rithm 4 has to terminate as well, as it does finite calls of a function that itself has upper bounded finitely many steps. □

5.1.2 Complexity

As Algorithm 3 is a subroutine of Algorithm 4, not all restrictions of Section 4.1.2 can be applied. The number of conflicts of leading positions within the matrix is one, but the general form of the matrix is unknown, as it is a subroutine of Algorithm 5 and Algorithm 5 might change the form of the matrix, but not the number of conflicts, as stated by Lemma 4.1.3.

Theorem 5.1.4
Considering a matrix $\mathbf{M} \in \mathbb{F}[x;\theta;\delta]^{\ell\times\ell}$ *with at most one conflict of leading positions over the Ore extension with* $\theta(a) = a^q$ *and* $\delta = 0$ *over some finite field* $\mathbb{F}_{q^m}$*, the elementary reduction operator Algorithm 3 has runtime complexity* $\mathcal{O}(\ell^2 \max_i\{\deg \mathbf{B}_i\})$.

Proof. As there is only one conflict, there will be at most $\ell - 1$ loops before the degree of the matrix will decrease or the matrix being in weak Popov form. This is due to Lemma 4.1.1, as a simple transformation that does not decrease the degree will have to lower the leading position. Lemma 4.1.3 states there will always be at most one leading position conflict. This conflict has to move to a lower positions every iteration, or lower the degree, as stated by Lemma 4.1.1. There can only be $\ell - 1$ different leading positions without the matrix being in weak Popov form, so there can only be $\ell - 1$ conflicting positions.

Besides checking for the correct form in Line 4, as well as the application of simple transformations on $\mathbf{B}$ in Line 12, everything is considered a free operation. The if statement as well as finding conflicting rows and fixing the leading position can be done in $\mathcal{O}(\ell)$ time, as one only has to search the last changed row, it might need $\mathcal{O}(\ell^2)$ preparation time. Extracting the degrees and coefficients is a matter of data structures and is therefore considered $\mathcal{O}(1)$.

Line 12 and Line 11 are applications of simple transformations on $\mathbf{B}$ and $\mathbf{U}$. For those, Lemma 4.1.2 states the complexity for the computation of one row element f of a simple transformation as $\mathcal{O}(L(f))$. Applying this on a row i results in $\mathcal{O}(\ell \max_j\{L(\mathbf{M}_{i,j})\})$.

For the simple transformation on $\mathbf{U}$ this can be further reduced, as it is created out of an identity matrix. A simple transformation can not involve the same two rows, as this would imply a decrease in degree. For the length of a polynomial in $\mathbf{U}$ to increase, one element has to be the target of at least two simple transformations.

$\mathbf{U}$ is an identity matrix at the beginning, this implies that for changing the same element by elementary row operations, the same row has to be involved twice as a source. As the leading position lowers every round, this is impossible. The maximum length of polynomials in $\mathbf{U}$ is therefore 1. This results in a complexity of $\mathcal{O}(\ell)$ for the simple transformation on $\mathbf{U}$.

As the length of elements in $\mathbf{B}$ might change during simple transformations, it can be upper bounded by the highest degree of an element in $\mathbf{B}$. The complexity can then be summed up as $\mathcal{O}(\ell \cdot (\ell \max_{i,j}\{\deg \mathbf{B}_{i,j}\} + \ell) + \ell^2)$ and the theorem follows. □

For the inductive reduction operator, the full restrictions will apply. As it utilises Algorithm 3, the complexity of Algorithm 4 will build upon Theorem 5.1.4. As matrix multiplication is a central operation in Algorithm 4, its complexity is important. As stated in Section 4.1.1, it is usually denoted by $\mathcal{O}(\ell^\omega)$ multiplications, where ω is understand to be $2 \leq \omega \leq 3$ and is usually implemented by the Strassen algorithm of [Str69] with $\omega \approx 2.8$.

Theorem 5.1.5
For some matrix $\mathbf{B} \in \mathbb{F}[x; a^q; 0]^{\ell \times \ell}$ obeying the introduced restrictions of decoding, the inductive reduction operator Algorithm 4 has runtime complexity $\mathcal{O}(t\ell^\omega \max_i \deg \mathbf{B}_i)$.

Proof. As Algorithm 3 reduced the degree by at least one, there are at most t loop passes and calls to Algorithm 3. For a single run, Theorem 5.1.4 gives a complexity of $\mathcal{O}(\ell^2 \max_i\{\deg \mathbf{B}_i\})$, and the proof states, that every returned $\mathbf{U}$ only contains monomials.

There are two matrix multiplications, $\mathbf{UB}$ and $R(\mathbf{UB})\mathbf{U}$ per loop. The multiplication by $\mathbf{U}$ does not need to be redone every loop pass, only the newly acquired result of a call to Algorithm 3 has to be applied. As a newly generated $\mathbf{U}$ contains only monomials, the multiplication complexity depends on the degrees of $\mathbf{B}$. A multiplication by a monomial can be done linearly in the length of the non–monomial polynomial.

An important remark here is that the linearly computation of the multiplication only holds for $\delta = 0$ and an automorphism θ that can be computed in $\mathcal{O}(1)$ regardless of the number of its applications. This is due to the multiplication having the form

$$\alpha x^\beta p(x) = \sum_{i=0}^{\deg p} \alpha \theta^\beta(p_i) x^{\beta+i}.$$

The condition check can be done linearly in the matrix size with a preparation time of $\mathcal{O}(\ell^2)$, as there is always only one conflict and a hidden data structure could

update this throughout calls to Algorithm 3.

This sums up to a complexity of

$$\mathcal{O}(\underbrace{\ell^2}_{\text{preparation time}} + t(\underbrace{\ell^2 \max_i \deg \mathbf{B}_i}_{\text{calls to } R(\mathbf{B})} + \underbrace{\ell^\omega \max_i \deg \mathbf{B}_i}_{\text{matrix multiplications}})),$$

which can be simplified to $\mathcal{O}(t\ell^\omega \max_i \deg \mathbf{b}_i)$, as long as $\omega \geq 2$. □

To generally compute a matrix in weak Popov form, it is sufficient to compute $R(\mathbf{B}, \Delta\mathbf{B})\mathbf{B}$, because the orthogonality defect has to be zero in weak Popov form and is based on the degree. For a matrix in the form of the Gao–like decoding matrix, it is sufficient to compute $R(\mathbf{B}, \deg \mathbf{B}_1 - \mathbf{v}_1)\mathbf{B}$, because of Lemma 4.1.4.

5.2 Accuracy Approximation of Polynomials

To speed up the algorithm, [Ale05] uses an approximation for polynomials. This mainly reduces computational complexity by shortening the polynomials that are multiplied, as the multiplication of polynomials depends on the number of terms.

Definition 5.2.1 (Accuracy Approximation)
The approximation to accuracy $t \in \mathbb{N}\backslash\{0\}$ for a polynomial p is

$$p|_t = \pi(p \cdot x^{t-\deg p-1})x^{\deg p+1-t}.$$

Where $\pi : \mathbb{F}[x, x^{-1}] \to \mathbb{F}[x]$ is the linear mapping defined by

$$\pi(x) = \begin{cases} x^k \longrightarrow x^k & : k \geq 0 \\ x^k \longrightarrow 0 & : k < 0 \end{cases}.$$

Informally, $p|_t$ reduces p to the first t terms. This includes terms with coefficient zero as long as their degree is strictly smaller than $\deg p$.

Accuracy approximation for $F[x; \theta; \delta]$ works as it does for regular polynomials since only regular multiplication occurs:

$$
\begin{aligned}
p|_t &= \pi(p \cdot x^{t-\deg p-1})x^{\deg p+1-t} \\
&= \pi(\sum_{i=0}^{\deg p} p_i x^i \cdot x^{t-\deg p-1})x^{\deg p+1-t} \\
&= \pi(\sum_{i=0}^{\deg p} p_i x^{i+t-\deg p-1})x^{\deg p+1-t} \\
&\overset{(*)}{=} \sum_{i=\deg p+1-t}^{\deg p} p_i x^{i+t-\deg p-1} \cdot x^{\deg p+1-t} \\
&= \sum_{i=\deg p+1-t}^{\deg p} p_i x^{i+t-\deg p-1+\deg p+1-t} \\
&= \sum_{i=\deg p+1-t}^{\deg p} p_i x^i.
\end{aligned}
$$

The transformation $(*)$ is due to

$$
\begin{aligned}
& i + t - \deg p - 1 \geq 0 && \text{(Condition of } \pi(x)\text{)} \\
\Leftrightarrow \quad & i \geq \deg p + 1 - t.
\end{aligned}
$$

This approximation can now be extended to vectors of polynomials using the degree of a vector.

Definition 5.2.2 (Accuracy Approximation for Vectors and Matrices) *Let* $\mathbf{v} = (p_1, p_2, \ldots, p_\ell) \in \mathbb{F}[x;\theta;\delta]^\ell$ *be a vector of Ore polynomials with* $\deg \mathbf{v} = \max_i\{\deg p_i\}$ *and polynomials* $p_i = \sum_{k=0}^{\deg p_i} p_{i,k}x^k$. *The accuracy approximation to a depth* t *is defined by*

$$\mathbf{v}|_t = (p_i = \sum_{k=\deg \mathbf{v}-t+1}^{\deg \mathbf{v}} p_{i,k}x^k).$$

Let $\mathbf{M} \in \mathbb{F}[x;\theta;\delta]^{m\times\ell}$ *be some matrix of polynomials. The accuracy approximation of depth* t $\mathbf{M}|_t$ *is defined by a row–wise application of the approximation:*

$$\mathbf{M}|_t = \begin{pmatrix} M_1|_t \\ M_2|_t \\ \vdots \\ M_m|_t \end{pmatrix}.$$

Informally, this does not reduce the polynomial to the first t terms, but up to t terms starting from the maximum degree within the vector. Example 5.2.1 gives an example for this.

Example 5.2.1 (Accuracy Approximation for Vectors and Matrices) *Take* $\mathbf{v} = (11x^7 + 9x^6 + 12x^4 + 9x, \quad 5x^6 + 2x^4 + 7x^2, \quad 12x^2 + 1)$ *as a vector, the degree of* $\mathbf{v}$ *is* $\deg \mathbf{v} = 7$*. The approximation to depth* $t = 8$ *would be the original vector. Some values for this vector are listed in Table 5.1.*

$t = 8$	$\mathbf{v}\|_8 = (11x^7 + 9x^6 + 12x^4 + 9x, \quad 5x^6 + 2x^4 + 7x^2, \quad 12x^2 + 1)$
$t = 7$	$\mathbf{v}\|_6 = (11x^7 + 9x^6 + 12x^4, \quad 5x^6 + 2x^4 + 7x^2, \quad 12x^2)$
$t = 2$	$\mathbf{v}\|_2 = (11x^7 + 9x^6, \quad 5x^6, \quad 0)$
$t = 1$	$\mathbf{v}\|_1 = (11x^7, \quad 0, \quad 0)$

Table 5.1: List of example approximations.

Let $\mathbf{M}$ *be the matrix*

$$\mathbf{M} = \begin{pmatrix} x^2 + 1 & 3x^2 & x^2 + x \\ 2x^5 & x^2 + x & 5x + 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

The following two approximations of depth $t = 1$ *and* $t = 4$ *shall be exemplary. Every row is treated separately:*

$$\mathbf{M}|_1 = \begin{pmatrix} x^2 & 3x^2 & x^2 \\ 2x^5 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{M}|_4 = \begin{pmatrix} x^2 & 3x^2 & x^2 + x \\ 2x^5 & x^2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

This approximation will allow improvements on the original algorithm.

5.3 Improvement

To calculate the matrices using the most suited approximation, Alekhnovich chose a recursive divide an conquer strategy when including the approximation. The resulting algorithm, altered for row–wise computation over a left Ore ring, is given by Algorithm 5 and called the improved recursive reduction operator.

It does as many calls to Algorithm 3 as Algorithm 4 does. Its performance gain stems from computing over shorter polynomials.

Algorithm 5: $\hat{R}(B,t)$, the improved recursive reduction operator.

Input: Basis $\mathbf{B} \in \mathbb{F}[x]^{m\times\ell}$
Output: Matrix $R(\mathbf{B}) \in \mathbb{F}[x]^{m\times m}$ so that either $R(\mathbf{B})\cdot\mathbf{B}$ is in weak Popov form or $\deg(R(\mathbf{B})\cdot\mathbf{B}) < \deg\mathbf{B}$.

```
let B = B|_t
if t = 1 then
    return R(B)
let U_1 = R̂(B, ⌊t/2⌋)
let B_1 = U_1 · B
return R̂(B_1, t − (deg B − deg B_1)) · U_1
```

5.3.1 Correctness

The proof of correctness is analogue to the original proof of correctness of [Ale05]. First, there are two identities that are useful for the proof of correctness.

Lemma 5.3.1
It holds for Algorithm 3 as $R(\mathbf{M})$, that

$$R(\mathbf{B}) = R(\mathbf{B}|_1).$$

Proof. Let $\mathbf{u}.\mathbf{v} \in \mathbb{F}[x;\theta;\delta]^\ell$ be rows of a matrix $\mathbf{B}$ with $\deg\mathbf{u} \geq \deg\mathbf{v}$ and the same leading position. A simple transformation can either reduce the degree or reduce the leading position of $\mathbf{u}$.

If the degree is reduced, the simple transformation is added as a matrix multiplication and the algorithm terminates. The result is the same for the algorithm with or without approximation, as the simple transformation only relies on the terms of maximum degree.

If the degree is not reduced, it holds, that:

$$(\mathbf{u} - \alpha x^\beta \mathbf{v})|_1 = (\mathbf{u}|_1 - \alpha x^\beta \mathbf{v}|_1)|_1,$$

because $\deg \alpha x^\beta\mathbf{v}$ is either zero or equal to $\deg\mathbf{u}$ and the transformation can not introduce any terms of degree $\deg\mathbf{u}$ during multiplication, as they are all strictly smaller than that. The additional approximation on the right side is necessary as Ore multiplication can introduce lower degree terms. It still implies the theorem, as the simple transformations are only computed using the highest degree in each row, it is therefore only important that no new terms of this degree are introduced.

The approximation will only change the outcome after a degree decreasing simple transformation, but the algorithm terminates in this case. □

This result can be extended to the inductive reduction operator.

Lemma 5.3.2
It holds for Algorithm 4 as $R(\mathbf{M}, t)$, that

$$R(\mathbf{B}, t) = R(\mathbf{B}|_t, t).$$

Proof. Lemma 5.3.1 implies that $R(\mathbf{B})$ will compute the same result for two matrices $\mathbf{A}, \mathbf{C}$, as long as $\mathbf{A}|_1 = \mathbf{C}|_1$.

Assume that for one $s \leq t$ it holds, that $\mathbf{A}|_s = \mathbf{C}|_s$. After one application of $R(\mathbf{A}) = R(\mathbf{C})$ on each matrix, the degree decreases by d and the approximation changes by d as well:

$$(R(\mathbf{A})\mathbf{A})|_{s-d} = (R(\mathbf{C})\mathbf{C})|_{s-d}.$$

This is true, because for some accuracy t it holds, that

$$\begin{aligned} x^{d_v - t - 1} x^{\beta = d_u - d_v} &= x^{d_u - d_v + d_v - t - 1} \\ &= x^{d_u - t - 1}. \end{aligned}$$

So a term not in scope before the approximation to depth t is not in scope if the new approximation is up to depth $t - d$ where $d = d_{u_{old}} - d_{u_{new}}$ is the difference in degree.

Algorithm 4 is an iteration over $R(\mathbf{B})$, so it computes $R(\mathbf{A}, s)$ for $s < t$.

As soon as the degree decreased by t, the algorithm stops and $R(\mathbf{B})$ is not called any further, so the resulting matrix can not diverge. □

Using these theorems, the correctness of the actual algorithm can be examined.

Theorem 5.3.1
Let $\mathbf{B} \in \mathbb{F}[x; \theta; \delta]^{m \times \ell}$ be some matrix over some Ore extension, then Algorithm 5 $\hat{R}(\mathbf{B}, t)$ computes a matrix $\mathbf{U} \in \mathbb{F}[x; \theta; \delta]^{m \times m}$ such that $\mathbf{UB}$ has $\deg \mathbf{UB} \leq \deg \mathbf{B} - t$ or $\mathbf{UB}$ is in weak Popov form.

Proof. As Algorithm 4 $R(\mathbf{M}, t)$ should be equivalent to Algorithm 5 $\hat{R}(\mathbf{M}, t)$, in combination with Theorem 5.1.2, it suffices to show that

$$\hat{R}(\mathbf{B}, t) = R(\mathbf{B}, t).$$

Using Lemma 5.3.1, it is clear for $t = 1$ that

$$\hat{R}(\mathbf{B}, 1) = R(\mathbf{B}|_1) = R(\mathbf{B}) = R(\mathbf{B}, 1).$$

The induction step for $t > 1$ will utilise the recursive definition:

$$\begin{aligned}
\hat{R}(\mathbf{B}, t) &= \hat{R}((\mathbf{U}_1\mathbf{B}|_t), t - (\deg \mathbf{B} - \deg \mathbf{U}_1\mathbf{B}|_t)) \cdot \mathbf{U}_1 \\
&= \hat{R}(\underbrace{\hat{R}(\mathbf{B}|_t, \left\lfloor \frac{t}{2} \right\rfloor)}_{\textit{Step to } \frac{t}{2}} \cdot \mathbf{B}|_t, t - \underbrace{(\deg \mathbf{B}|_t - \deg(\hat{R}(\mathbf{B}|_t, \left\lfloor \frac{t}{2} \right\rfloor) \cdot \mathbf{B}|_t)))}_{\textit{already achieved degree reduction}} \cdot \underbrace{\hat{R}(\mathbf{B}|_t, \left\lfloor \frac{t}{2} \right\rfloor)}_{\textit{Step to } \frac{t}{2}} \\
&\overset{IS}{=} R(R(\mathbf{B}|_t, \left\lfloor \frac{t}{2} \right\rfloor) \cdot \mathbf{B}|_t, t - (\deg \mathbf{B}|_t - \deg(R(\mathbf{B}|_t, \left\lfloor \frac{t}{2} \right\rfloor) \cdot \mathbf{B}|_t))) \cdot R(\mathbf{B}|_t, \left\lfloor \frac{t}{2} \right\rfloor) \\
&\overset{(1)}{=} R(\mathbf{B}|_t, t) \overset{(2)}{=} R(\mathbf{B}, t)
\end{aligned}$$

Steps (1) and (2) are crucial. Step (2) is an application of Lemma 5.3.2.

Step (1) utilises the behaviour of Algorithm 4 to compute the reductions by one consecutively. For any k with $0 < k \leq t$, Algorithm 4 first computes a reduction by k and then by $t - k$. The reduction matrices are then combined. As matrix multiplication is associative, the order is of no importance:

$$\begin{aligned}
R(\mathbf{B}, t) &= \mathbf{U} \\
&= \underbrace{\underbrace{\mathbf{T}_t\mathbf{T}_{t-1}\dots\mathbf{T}_{k+1}}_{t-k \textit{ many}}}_{=\mathbf{W}_{t-k}} \underbrace{\underbrace{\mathbf{T}_k\mathbf{T}_{k-1}\dots\mathbf{T}_2\mathbf{T}_1}_{k \textit{ many}}}_{=\mathbf{W}_k} \\
&\overset{(*)}{=} \mathbf{W}_{t-k}\mathbf{W}_k \\
&= R(R(\mathbf{B}, k)\mathbf{B}, t - k)R(\mathbf{B}, k).
\end{aligned}$$

Note that the term $\mathbf{W}_{t-k}$ introduced in step $(*)$ is a place holder for the product $\mathbf{T}_t \dots \mathbf{T}_{t-k+1}$ and not identical to the matrix $\mathbf{T}_{t-k}$ of the previous step. If $\mathbf{T}_i$ transformed $\mathbf{T}_{i-1} \dots \mathbf{T}_1\mathbf{B}$ into weak Popov form, it would follow for all $j > i$ that $\mathbf{T}_j = I$ is the identity matrix.

This induction proves the theorem. □

As the algorithm is deemed correct, its complexity is of interest.

5.3.2 Complexity

The complexity of Algorithm 5 will be examined in context of decoding, as this allows for much more fine grained analysis. The complexity of general reduction using the

algorithm will be worse as the assumptions made, for example no introduction of lower degree terms during simple transformation, would not hold.

Theorem 5.3.2
Let $2 \leq \omega \leq 3$ be the number indicating complexity of matrix multiplication. Under the restrictions proposed in Section 4.1.2, Algorithm 5 has a runtime complexity of $\mathcal{O}(\ell^\omega M(t) \log t)$.

Proof. The complexity will be split in the complexity of all calls to Algorithm 3 and the rest of the algorithm. The complexity of the approximation will not be considered. Actual storage of the polynomials is implementation dependent, but an example implementation using a list or array to store coefficients, could be modified to ignore coefficients after t entries.

The accuracy approximation of depth t ensures that the polynomials in $R(B)$ can not get longer. As the length is one, this reduces the complexity of Algorithm 3 not only to $\mathcal{O}(\ell^2 \max_{i,j}\{L(B_{i,j})\})$, but to $\mathcal{O}(\ell^2)$. The complexity for at most t calls to Algorithm 3 is therefore $\mathcal{O}(t\ell^2)$.

Further let $f(t)$ describe the complexity of Algorithm 5 without the calls to Algorithm 3. As Algorithm 5 does two recursive calls that add up to t of size around $\frac{t}{2}$ this can be modelled by two calls of size $\frac{t}{2}$. It holds, that

$$f(t) \leq \mathcal{O}(\ell^\omega M(t)) + 2f\left(\frac{t}{2}\right).$$

This expands until $f(1)$ is called as a base case. This is the case as soon as

$$\begin{aligned} & 1 \geq \frac{t}{2^x} \\ \Leftrightarrow \quad & 2^x = t \\ \Leftrightarrow \quad & x = \log t. \end{aligned}$$

This could be verified using the master theorem of computational complexity.

So $f(t)$ can be estimated as

$$f(t) \leq \mathcal{O}(\ell^\omega M(t) \log t).$$

The combined complexity of Algorithm 5 can be summed up as $\mathcal{O}(\ell^\omega M(t) \log t + \ell^2 t + \ell^2 t \log t)$ and, assuming $M(t) \geq t$ and $\omega \geq 2$, can be simplified to $\mathcal{O}(\ell^\omega M(t) \log t)$. □

Using the introduced μ and the complexity for multiplication, the following theorem compiles the analysis for comparison with earlier findings.

Theorem 5.3.3
There is an algorithm to compute the weak Popov form of a matrix abiding Section 4.1.2 in $\mathcal{O}(\ell^\omega \mu^{1.69} \log \mu)$ *with* $\mu = \max_i\{\deg \mathbf{B}_{i,i} - \mathbf{v}_i\}$.

Proof. To compute the weak Popov form of a matrix $\mathbf{B} \in \mathbb{F}[x; a^q; 0]^{\ell \times \ell}$ using Alekhnovich's algorithm the computation $\hat{R}(\mathbf{B}, \Delta\mathbf{B})\mathbf{B}$ is necessary, where $\hat{R}$ is Algorithm 5. The post processing matrix multiplication can be done in $\mathcal{O}(\ell^\omega M(\mu))$ as μ limits the length of the polynomials.

The invocation of Algorithm 5 can be transformed to $\hat{R}(\mathbf{B}, \mu - \mathbf{v}_1)$ because of Theorem 4.1.2, which results in a complexity of $\mathcal{O}(\ell^\omega M(\mu - \mathbf{v}_1) \log(\mu - \mathbf{v}_1))$. Multiplication can be done in $\mathcal{O}(n^{1.69})$ as stated in [WZ13, Section 3.1.3].

As $\mathbf{v}_1 \in \mathbb{N}_0$, the resulting complexity is $\mathcal{O}(\ell^\omega \mu^{1.69} \log \mu)$. □

5.4 Post–Computation

As Alekhnovich's algorithm only computes a matrix that can transform the base matrix into weak Popov form, a matrix multiplication is necessary afterwards. This implies an additional factor of $\mathcal{O}(\ell^\omega M(n))$ with n the largest degree within the matrix.

This step can be reduced to a vector-matrix multiplication for decoding. While this has no implication on the overall asymptotic complexity of the algorithm, in practice this might speed up the decoding process.

Theorem 5.4.1
Using information of within the algorithm, the complexity to compute the solution vector for decoding after applying Alekhnovich's algorithm is $\mathcal{O}(\ell^2 M(n))$.

Proof. The only row of interest is the row with leading position in the first column, as every other row will fail the degree conditions. For decoding of the form of Section 4.1.2, the matrix is in weak Popov form if and only if a leading position switches to column one.

This is true, as the matrix contains only one conflict of leading position. No leading position is in position one, the first column, as for the row i with $2 \leq i \leq \ell$ the

leading position is, in the beginning, on the diagonal:

$$LP(\mathbf{B}_i) = i.$$

To remain in conflict after a simple transformation, the target of the previous simple transformation has to have a leading position different from one, as all other leading positions are still different from one. If a leading position equals to one, the single conflict has to be resolved.

The last operation will result in this resolution, as the algorithm will terminate afterwards. The target row of the last operation is the row of interest, as it has leading position one. Retrieving this information can be done without impact on computational complexity.

Let i be the row of interest. Computing $(\mathbf{UB})_i$ is equivalent to computing $\mathbf{U}_i\mathbf{B}$ which is only a multiplication of a vector by a matrix:

$$\mathbf{z} = (\mathbf{z}_j = \sum_{k=1}^{\ell} \mathbf{U}_{i,k}\mathbf{B}_{k,j}).$$

This can be done in $\mathcal{O}(\ell^2 M(n))$. □

5.5 Summary

This chapter converted Alekhnovich's algorithm to general non–commutative polynomials. This allows the application of the algorithm to decode Gabidulin codes. An analysis of its performance on decoding resulted in a runtime of $\mathcal{O}(\ell^\omega n^{1.69} \log n)$, where ω is the complexity of matrix multiplication. It can be assumed that $\omega = 2.8$.

The next chapter will examine the special case of non–interleaved decoding using Alekhnovich's algorithm.

6 Regular Decoding Using Alekhnovich's Algorithm

For non–interleaved decoding of Gabidulin codes, Theorem 3.2.3 gives a solution matrix of fixed size $\ell = 2$. This chapter will examine the results of Chapter 5 in the context of non–interleaved decoding.

6.1 Sub–Quadratic Decoding

For a complete decoding process, the complexity of module minimisation is not sufficient, see Section 3.3. For actual decoding the following steps are necessary:

1. Interpolation of the received word r to get $\hat{r}$.
2. Solving the key equation using module minimisation.
3. Retrieval of the information polynomial by division.

These steps are more formalised in Algorithm 6.

Algorithm 6: $decode(\mathcal{C}_\mathcal{G}, r)$, Modified [WZ13, Algorithm 3.6].

Input: Gabidulin code $\mathcal{C}_\mathcal{G}$ and a received word r
Output: Information polynomial f or decoding failure.

```
let r̂ be the interpolation of r
let (λ, ψ) be the row with leading position one calculated using Alekhnovich
let (f, remainer) = LeftDivision(λ, ψ)
if remainer = 0 then
    return f
else
    return "Decoding failure"
```

The minimal subspace polynomial can be precomputed and does not change, so the complexity of calculating it is not relevant.

Interpolation

The first step, interpolation, as described in Definition 3.2.4, provides a limit of complexity of interpolation.

Theorem 6.1.1
The complexity of interpolation is at most $\mathcal{O}(n^2)$.

Proof. All minimal subspace polynomials can be precomputed. The only relevant computation to interpolate a received word is multiplication by r_i and summation of all terms. The additions are ignored, as they are dominated by complexity of multiplication.

There are n multiplications by a factor $r_i \in \mathbb{F}_{q^m}$ to be done. A multiplication of this sort has a complexity of $\mathcal{O}(\deg_q L)$, where $L \in \mathcal{L}_q[x]$ is the function being multiplied on. L is a minimal subspace polynomial and has, according to [WZ13, Lemma 2.9], a degree of $\deg_q L = dim(\mathcal{U})$, where $\mathcal{U}$ is the space of the minimal subspace polynomial $L = M_{\mathcal{U}}$.

The dimension of $\langle \mathcal{G} \backslash g_i \rangle$ is $n-1$ as all g_i are linear independent. This results in a degree $\deg_q L = n-1$ and a complexity of $\mathcal{O}(n(n-1)) = \mathcal{O}(n^2)$ multiplications over $\mathbb{F}_{q^m}$ for n multiplications. □

[WZ13] uses special properties of certain bases to compute the interpolation using the q-transform and its inverse. Although not possible for all parameters, as n has to divide m, this allows a computation of the interpolation using the q-transform.

This can be done in $\mathcal{O}(n^3)$ over $\mathbb{F}_q$ instead of $\mathbb{F}_{q^m}$, due to the ext transformation. Even though the reverse is not true, a linear operation in $\mathbb{F}_{q^m}$ can be computed in $\mathcal{O}(n^3)$, using the ext mapping, operations over $\mathbb{F}_q$. This can be considered sub–quadratic computation of the interpolation and is used in Line 1.

Further research showed the possibility of computing the q-transform using a Hankel matrix, resulting in quasi linear time complexity for interpolation. See [MRT05] as a Hankel matrix is similar to Toeplitz matrix, because a Hankel matrix $\mathbf{H}$ can be written in the form $\mathbf{H} = \mathbf{JT}$ for a Toeplitz matrix $\mathbf{T}$ and a matrix of the form:

$$\mathbf{J} = \begin{pmatrix} 0 & 0 & 1 \\ \vdots & \iddots & \vdots \\ 1 & 0 & 0 \end{pmatrix}.$$

Solving the Key Equation

Solving the key equation can be done using module minimisation. The process of module minimisation is explored in Chapter 4. This resulted in Alekhnovich's algorithm with complexity $\mathcal{O}(\ell^{\omega}\mu^{1.69}\log\mu)$, which is used in Line 2. For $\ell = 2$ this results in a complexity of $\mathcal{O}(n^{1.69}\log n)$ for the length of a codeword n.

Theorem 6.1.2
It holds, that

$$n^{1.69}\log n = o(n^2).$$

Proof. For $f = o(g)$ to be true, it is sufficient to show

$$\lim_{x\to\infty}\left|\frac{f}{g}\right| = 0.$$

In this case it is $f = n^{1.69}\log n$ and $g = n^2$. The limit can be calculated in the following way:

$$\begin{aligned}
\lim_{n\to\infty}\left|\frac{n^{1.69}\log n}{n^2}\right| &= \lim_{n\to\infty}\left|\frac{\log n}{n^{0.31}}\right| \\
&\overset{(*)}{=} \lim_{n\to\infty}\left|\frac{\frac{d}{dn}\log n}{\frac{d}{dn}n^{0.31}}\right| \\
&= \lim_{n\to\infty}\left|\frac{n^{-1}\log^{-1}(2)}{n^{-0.69}}\right| \\
&= \lim_{n\to\infty}\left|\frac{1}{n^{0.31}\log(2)}\right| \\
&= 0.
\end{aligned}$$

Step $(*)$ is an application of l'Hôpital's rule as $\lim_{n\to\infty} n^{0.31} = \infty = \lim_{n\to\infty}\log n$. □

Therefore, an algorithm of complexity $\mathcal{O}(n^{1.69}\log n)$ can be called sub–quadratic.

Division

For division there have been efforts to reduce division for skew polynomials to multiplication. While [WZ13] gives a complexity of $\mathcal{O}((n-m)m)$, which is quadratic, [CL12, Section 2.1.2] describes an algorithm reducing the division to the complexity

of multiplication. As multiplication can be done in $\mathcal{O}(n^{1.69})$, this implies a sub–quadratic division. Using this result in Line 3 results in an algorithm of the form of Algorithm 6, that can be considered sub–quadratic.

6.2 Importance

Even though Alekhnovich's algorithm on its own does not yield a sub–quadratic decoding algorithm, it solves an important step of the decoding algorithm in sub–quadratic complexity considering runtime. Both other steps are actively researched and quite possibly solvable, or can be considered solved, in sub–quadratic time. This would imply a sub–quadratic decoding algorithm for decoding of regular Gabidulin codes.

7 Conclusion

This thesis mainly considered a conversion of Alekhnovich's algorithm of [Ale05]. The algorithm performs well over Ore extensions with few adaptations. Table 7.1 gives a short comparison of the algorithms that have been converted for module minimisation over Ore extensions by [LNPS15] and Alekhnovich's variant of this thesis.

In this table the parameter ℓ describes the size of the matrix, in context of decoding, ℓ measures the size of the interleaving. The parameter n is a measure of the degree of the involved polynomials or the size of the codewords. The parameter ω describes the complexity of matrix multiplications. A naive implementation uses $\omega = 3$, while there are known algorithms for $\omega \approx 2.374$ (cf. [CW90]), in practice it can be assumed to be $\omega \approx 2.8$ (cf. [Str69]).

Algorithm		Complexity
Mulders–Storjohann	Section 4.2	$\mathcal{O}(\ell^2 n^2)$
Demand–Driven	Section 4.3	$\mathcal{O}(\ell n^3)$ (Special Case $\mathcal{O}(\ell n^2)$)
Alekhnovich	Chapter 5	$\mathcal{O}(\ell^\omega n^{1.69} \log n)$

Table 7.1: Comparison of the runtime of the converted algorithms for decoding over non–commutative polynomial matrices, where $2 \leq \omega \leq 3$ describes the complexity of matrix multiplication.

While the demand–driven algorithm is a clear improvement over the classic Mulders–Storjohann algorithm for the special case described in Section 4.3, it offers an $\ell - \mu$ trade–off. Alekhnovich's algorithm on the other hand, offers a $\mu - \ell$ trade–off. For non–interleaved decoding Alekhnovich's algorithm implies a sub–quadratic algorithm to solve the key equation, as

$$n^{1.69} \log n = o(n^2).$$

This gives an important step for feasible sub–quadratic decoding of Gabidulin codes.

Further Research

This thesis only highlights the step of module minimisation and does not touch on the complexity of interpolation, root finding or other steps. There are different approaches, aside from module minimisation, to decoding of Gabidulin codes, which are not included and compared for their complexity.

As Alekhnovich's algorithm depends on the complexity of multiplication, its complexity would improve if faster methods of multiplication could be applied. The full decoding would profit from a fast division algorithm.

At last an implementation of the algorithms in this thesis and an analysis of its real world application, would be an important step to take.

Notations

Notation	Summary
q	Power of a prime.
$[i] = q^i$	q-power for some integer i.
$\mathbb{F}_q$	Finite field of order q.
$\mathbb{F}_{q^m}$	Extension field of a finite field of order q of degree m.
$\mathbb{F}_q^m$	Space of dimension m over a finite field of order q.
$F[x;\theta;\delta]$	Ore extension of field F with derivative δ.
$\mathcal{L}_q[x]$	Ring of linearised polynomials (see definition 2.2.1).
$c = (c_1, \ldots, c_n)$	Codeword of length n.
$e = (e_1, \ldots, e_n)$	Error word of length n.
$r = (r_1, \ldots, r_n) = c + e$	Received word of length n.
$\mathcal{E} = \{i : e_i \neq 0\}$	Set of error positions.
Λ	Error locator or error span polynomial.
λ	A solution candidate for Λ.
$\Omega = \Lambda f$	Abstraction of the right hand side of equation 3.3.
ψ	A solution for Ω.
$\mathbf{M}$	A matrix M.
$\mathbf{M}_i$	The i-th row of some matrix M.
$\mathbf{M}_{i,j}$	The j-th element of the i-th row of some matrix M.

Bibliography

[Ale02] M. Alekhnovich. Linear diophantine equations over polynomials and soft decoding of reed-solomon codes. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 439–448, 2002.

[Ale05] Michael Alekhnovich. Linear diophantine equations over polynomials and soft decoding of reed-solomon codes. volume 51, pages 2257–2265, July 2005.

[Art91] Michael Artin. *Algebra*. Prentice Hall, 1991.

[Ber84] Elwyn R. Berlekamp. *Algebraic Coding Theory, Revised Edition*. Aegean Park Press, 1984.

[BL05] Thierry Berger and Pierre Loidreau. Designing an efficient and secure public-key cryptosystem based on reducible rank codes. In Anne Canteaut and Kapaleeswaran Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 218–229. Springer Berlin Heidelberg, 2005.

[Bos13] Martin Bossert. *Kanalcodierung*. Oldenbourg Verlag, third edition, 2013.

[CK91] David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, July 1991.

[CL12] X. Caruso and J. Le Borgne. Some algorithms for skew polynomials over finite fields. *ArXiv e-prints*, December 2012.

[Coh95] Paul Moritz Cohn. *Skew Fields: Theory of General Division Rings*. Cambridge University Press, July 1995.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990. Computational algebraic complexity editorial.

[Gab85] Ernst M. Gabidulin. Theory of Codes with Maximum Rank Distance. *Probl. Peredachi Inf.*, 21(1):3–16, 1985.

[Gao03] Shuhong Gao. A new algorithm for decoding reed-solomon codes. In VijayK. Bhargava, H.Vincent Poor, Vahid Tarokh, and Seokho Yoon, editors, *Communications, Information and Network Security*, volume 712 of *The Springer International Series in Engineering and Computer Science*, pages 55–68. Springer US, 2003.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[HMK+06] Tracey Ho, M. Medard, R. Koetter, D.R. Karger, M. Effros, Jun Shi, and B. Leong. A random linear network coding approach to multicast. *Information Theory, IEEE Transactions on*, 52(10):4413–4430, Oct 2006.

[ISO06] International Organization for Standardization ISO. Information technology — Automatic identification and data capture techniques — QR Code 2005 bar code symbology specification. Standard, International Organization for Standardization, Geneva, CH, March 2006.

[KK08] Ralf Koetter and Frank R. Kschischang. Coding for errors and erasures in random network coding. 54(8):3579–3591, 2008.

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, 1996.

[LNPS15] Wenhui Li, Johan S. R. Nielsen, Sven Puchinger, and Vladimir Sidorenko. Solving shift register problems over skew polynomial rings using module minimisation. 2015.

[MRT05] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A fast algorithm for the inversion of general Toeplitz matrices. *Computers & Mathematics with Applications*, 50(5):741–752, 2005.

[MS74] George Marsaglia and George P. H. Styan. Equalities and inequalities for ranks of matrices. *Linear and Multilinear Algebra*, 2(3):269–292, 1974.

[MS03] Thom Mulders and Arne Storjohann. On lattice reduction for polynomial matrices. *Journal of symbolic computation*, 35(4):377–401, 2003.

[Nie13a] Johan S. R. Nielsen. Generalised Multi-sequence Shift-Register synthesis using module minimisation. *2013 IEEE International Symposium on Information Theory*, pages 882–886, July 2013.

[Nie13b] Johan S. R. Nielsen. Power Decoding of Reed-Solomon Codes Revisited. *CoRR*, abs/1311.1, November 2013.

[Ore33] Oystein Ore. Theory of Non-Commutative Polynomials. *Annals of Mathematics*, 34(3):480–508, 1933.

[Ros94] Jonathan Rosenberg. *Algebraic K-Theory and Its Applications*. Springer Science & Business Media, June 1994.

[Rot06] Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, New York, NY, USA, 2006.

[SSB06] G. Schmidt, V. Sidorenko, and M. Bossert. Decoding reed-solomon codes beyond half the minimum distance using shift-register synthesis. In *Information Theory, 2006 IEEE International Symposium on*, pages 459–463, July 2006.

[SSB09] G. Schmidt, V.R. Sidorenko, and M. Bossert. Collaborative decoding of interleaved reed–solomon codes and concatenated code designs. *Information Theory, IEEE Transactions on*, 55(7):2991–3012, July 2009.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[Wic94] Stephen B. Wicker. *Reed-Solomon Codes and Their Applications*. IEEE Press, Piscataway, NJ, USA, 1994.

[WZ13] Antonia Wachter-Zeh. *Decoding of Block and Convolutional Codes in Rank Metric*. PhD thesis, 2013.