Technical University of Denmark

Interpolation and List Decoding of Algebraic Codes

Brander, Kristian; Beelen, Peter ; Høholdt, Tom

Publication date:
2010

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Brander, K., Beelen, P., & Høholdt, T. (2010). Interpolation and List Decoding of Algebraic Codes. Kgs. Lyngby, Denmark: Technical University of Denmark (DTU).

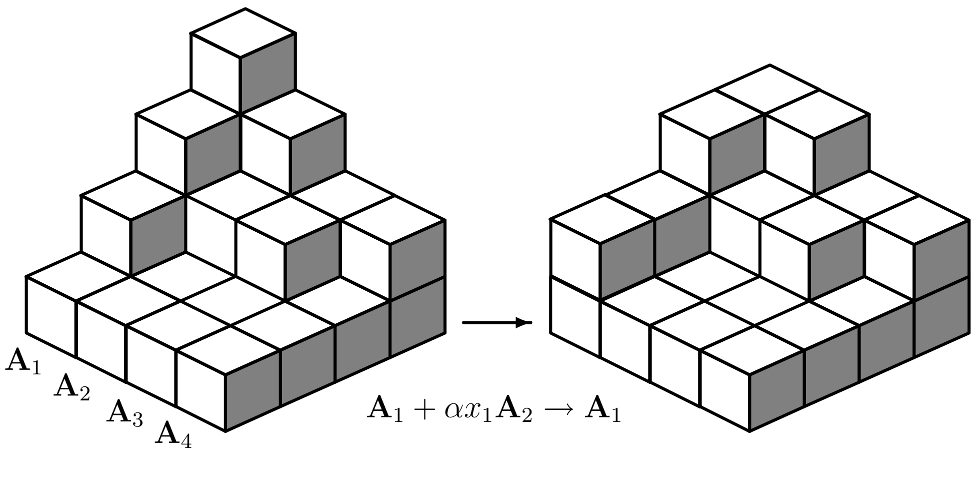
Interpolation and List Decoding of Algebraic Codes

PhD thesis

Kristian Brander

Thesis advisors: Peter Beelen and Tom Høholdt

January 28, 2010



DTU Mathematics
Department of Mathematics

Date Kristian Brander

Technical University of Denmark
Department of Mathematics
Matematiktorvet
Building 303S
DK-2800 Kgs. Lyngby
Denmark
Phone: +45 4525 3031
Fax: +45 4588 1399
`instadm@mat.dtu.dk`

Summary

Interpolation and List Decoding of Algebraic Codes

Error-correcting codes are used in digital communication and storage systems to prevent the influence of external noise from corrupting data. Examples of such systems are CD and DVD players, satellite communication systems and the transatlantic optical fiber cables that constitute an important link in the Internet's infrastructure.

A practical error-correcting code must provide a good trade-off between communication rate and error-correcting capability, and furthermore it should be efficiently decodable. List decoding is a flexible scheme for decoding error-correcting codes, and recent results has shown that it allows for *theoretically optimal* use of communication channels. The error-correcting codes and decoders in these results are completely explicit. Despite this fact it remains a difficult challenge to take advantage of the promised optimality in practical implementations. This fact is due to the lack of efficient list decoders capable of operating at the rate at which modern digital communication take place.

This thesis investigates list decoders for various algebraic error-correcting codes with emphasis on the efficiency with which these can be implemented. An integral part of the studied list decoders is a constrained multivariate interpolation problem. In the thesis a fast method for solving this problem is developed. Furthermore, a new method for decoding the theoretically optimal error-correcting codes mentioned above is given, and its virtues and limitations are explored.

Resumé

Interpolation og Listeafkodning af Algebraiske Koder

Fejlrettende koder anvendes i digitale kommunikations- og lagringssystemer til at beskytte data mod at blive ødelagt af udefrakommende støj. Eksempelvis anvendes fejlrettende koder i CD- og DVD-afspillere, satellitkommunikationssystemer og i de transatlantiske optiske fiberkabler der er en vigtig del af internettets infrastruktur.

En central egenskab ved en fejlrettende kode, er dens afvejning af kommunikationshastighed og fejlrettende egenskaber. Ydermere skal en god fejlrettende kode kunne afkodes effektivt. Listeafkodning er en fleksibel metode til at afkode fejlrettende koder og det er for nyligt blevet vist, at man med anvendelse af listeafkodning kan opnå *teoretisk optimal* udnyttelse af en kommunikationskanal. De fejlrettende koder og listeafkodere, der indgår i dette resultat, er beskrevet fuldstændig eksplicit. På trods af dette forbliver det en udfordring at omsætte den teoretiske optimalitet til en praktisk anvendelig implementering. Dette skyldes, at der endnu ikke er udviklet listeafkodere, der kan følge med i det tempo, hvormed moderne digital kommunikation foregår.

I nærværende afhandling undersøges listeafkodere for forskellige typer algebraiske fejlrettende koder med særligt fokus på den effektivitet der kan opnås i praktiske implementeringer af disse. Et vigtigt trin i alle de undersøgte listeafkodere består i at løse et særligt flervariabelt interpolationsproblem. I afhandlingen udvikles en effektiv metode til at løse netop denne type problemer. Der udvikles ydermere en ny metode til at afkode de ovennævnte teoretisk optimale fejlrettende koder, og denne metodes fordele og ulemper indkredses.

Preface

This thesis is submitted in partial fulfilment of the requirements for acquiring the PhD degree. The thesis describes the work carried out by the author while he was a PhD student at the Department of Mathematics, Technical University of Denmark, from January 2007 to January 2010. The work was funded by the Technical University of Denmark and supervised by Associate Professor Peter Beelen and Professor Tom Høholdt, both employees at the Technical University of Denmark.

The thesis is based on, and in many cases an extension of, the manuscripts:

- P. Beelen and K. Brander. Key-equations for list decoding of Reed–Solomon codes and how to solve them. Accepted for publication in Journal of Symbolic Computation, November 2008.
- P. Beelen and K. Brander. Efficient list decoding of a class of algebraic-geometry codes. Submitted to Advances in Mathematics of Communications, November 2009.
- P. Beelen and K. Brander. *Gröbner Bases, Coding, and Cryptography*, chapter: Decoding Folded Reed–Solomon Codes Using Hensel–Lifting, pages 389–394. Springer-Verlag, 2009.

Acknowledgements

I would like to thank my advisor Peter Beelen for guidance and support. It has been a great pleasure and privilege to have had such an insightful mentor, inspirator and moderator during the last three years. I would also like to sincerely thank my second advisor Tom Høholdt for introducing me to the subject of error-correcting codes, for always supporting and encouraging me and generally for being such a genuinely nice person.

I am deeply grateful to Madhu Sudan for hosting my stay at MIT in the spring semester 2009. From him I have learned most of what I know about the probabilistic nature of things, and about the joys of raw fish. I would also like to thank Swastik Kopparty for enlightening conversations and for sharing some of his broad knowledge.

I would like to thank Jørn Justesen for many interesting conversations. Whether academic or of a 'broader' nature they have always been a pleasure to me. Also, thanks, Søren for always being good company and for having good taste in music and coffee.

Finally, thank you, Tilde for being so wonderfully non-mathematical.

Kristian Brander
Kgs. Lyngby, January 2010

Contents

Chapter 1

Introduction

In this thesis we will investigate the role of multivariate polynomial interpolation in list decoding of algebraic error-correcting codes. Our focal point will be Sudan's seminal paper *Decoding of Reed–Solomon Codes beyond the Error-Correction Bound* [65], which not only gave the first explicit and efficient list decoder, but also connected the theory of error-correcting codes with the subject of complexity theory. Since then, list decoding has been developed in a fruitful interplay between the two fields, and has found numerous applications in both.

1.1 List decoding

In this section we will introduce list decoding and put it into its information theoretic context.

1.1.1 Error-correcting codes and channels

Error-correcting codes were invented to answer the question: Say we have some means for communication, a *channel*, capable of sending symbols from a finite alphabet Σ, in such a way that whenever the sender transmits n symbols, at most τ of them are corrupted when they arrive at the receiver. How do we communicate over such a channel, i.e. how can a sender make sure that an error-free copy of a message is made available at the receiver end? This problem arises in a number of scenarios of which we now mention a few.

1. Introduction

- A data harvesting satellite needs to transmit its data to a control centre on Earth. It does so via radio waves, but due to the physical interference that affects the waves as they travel through space, the data may be (and most likely is) corrupted when it arrives to Earth.
- A musician wishes to communicate his or hers latest piece to an audience. With the help of a record company, the track is put on CDs and made available in record stores. The CDs are the musician's communication channels. With time the plastic in CDs degenerate and if furthermore the CDs have been treated less considerately, the message that gets through to the listener is a distorted version of the original track.
- Most digital communication generated by internet traffic between USA and Europe is transferred through optic fibers lying on the bottom of the Atlantic Ocean. Due to deterioration of the fiber material, this channel inflicts errors on the messages it transmits. In fact at the rate with which the fibers are currently used, as many as 10^8 errors may occur per second [36].

If a communication channel can corrupt *all* symbols of a message arbitrarily, then no information gets through from the sender to the receiver. On the other hand, if $\tau/n < 1$ and n is sufficiently large, then the fundamental theorem of information theory, Shannon's Channel Coding Theorem [61], states that the sender can actually convey some information to the receiver via the channel. This promise can be realized with the use of error-correcting codes.

The Hamming distance of two vectors $\mathbf{x}$ and $\mathbf{y}$ in Σ^n, is defined as

$$d_H(\mathbf{x}, \mathbf{y}) = |\{i \mid x_i \neq y_i\}|.$$

With this notion we can formulate the properties of the channel described above as follows: if $\mathbf{x}$ is the transmitted word and $\mathbf{y}$ is the word that the receiver gets, then $d_H(\mathbf{x}, \mathbf{y}) \leq \tau$. An error-correcting code (or simply a code) C of length n defined over the alphabet Σ, is a collection of points in Σ^n. The *dimension* of a code C is

$$k = \log_{|\Sigma|}(|C|),$$

and the *rate* of C is defined as the "density" of C in Σ^n, i.e. as $R = k/n$. An error-correcting code C can be used in the channel communication problem outlined above, by restricting the sender to only transmit vectors from C, rather than from all of Σ^n. If $R < 1$ then this is a real restriction which gives some a priori information about the transmitted words. For a given received word, the receiver may attempt to use this information to recover from the errors imposed by the channel, and this process is known as *decoding*. The minimum distance of a code is defined as

$$d(C) = \min\{d_H(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\}.$$

The minimum distance is an important parameter of a code, determining how many errors the receiver's decoding procedure can correct.

1.1.2 Unique decoding

If $d(C) > 2\tau$, then the receiver can recover a transmitted word $\mathbf{x} \in C$ from the corresponding received word $\mathbf{y}$ as the *unique* vector in C within Hamming distance at most τ from $\mathbf{y}$, see Figure 1.1. Thus in this case, the sender can transmit one of the $|C|$ possible messages by using the channel n times, and the receiver is guaranteed to be able to recover an error-free copy of this message. When the channel is used like this, the fraction of each transmission that actually carries information is the rate R. The so-called Singleton bound states that $d(C)/n \leq 1 - R + 1/n$, and furthermore codes achieving this bound exist [46, Chap. 11]. Therefore we can summarize the above discussion by saying that if $\tau/n < \frac{1}{2}(1 - R)$, then it is possible to communicate with rate R on the channel.

1.1.3 List decoding and list decoders

If $d(C) \leq 2\tau$, then we can not in general expect that there is a unique codeword within distance τ from the received word and thus in this case, we are not guaranteed to uniquely recover the transmitted word, see Figure 1.2. However if τ is not too much larger than $d(C)/2$, then it may still be the case that only a small *list* of codewords are within distance τ of the received word. In this case the receiver is not necessarily able to pin down the transmitted word exactly, but knowing that this word can be found among a small list of words, may still be a lot more useful to the receiver, than just not getting any information at all. For instance:

- If the transmitted message is a picture, or some other structured data, then it is possible that the receiver can use this extra structure to decide which of the codewords on the list, is the correct one.

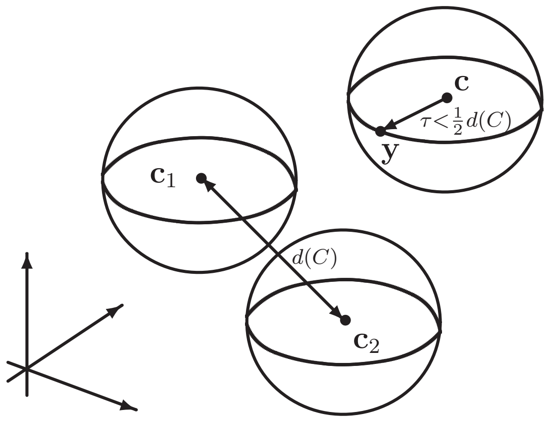


Figure 1.1: The Hamming ball with center $\mathbf{c}$ and radius τ is $B_\Sigma(\mathbf{c}, \tau) = \{\mathbf{x} \in \Sigma^n \mid d_H(\mathbf{c}, \mathbf{x}) \leq \tau\}$. If $\tau < d(C)/2$ then the balls of radius τ around the codewords in C do not overlap.

- It may happen that there is only one codeword on the list. Indeed, this is often the case for random errors (see e.g. [48, p. 33] or [55]). In fact, for a given code, proving the existence of a received word with many nearby codewords is in general a difficult combinatorial challenge [9, 37].
- If the receiver insists on obtaining a single guess on the transmitted word, then he can select some codeword which is closest to the received word, among all the codewords on the list. This approach corresponds to *maximum likelihood decoding*, since the closest codeword is also the most probable one.

The process of finding a small list of codewords within distance τ from a received word is called *list decoding*. This notion was introduced independently by Elias [18] and Wozencraft [68] in the late 1950's. It is important to formalize what a *small* list is, since if we allow too large lists, then list decoding becomes trivial (e.g. if the decoder is allowed to output all the words of C).

Definition 1.1 (List decoder)**.** Let C be an error-correcting code of length n, and let τ be an integer. A τ–list decoder for C is an algorithm $\mathcal{A}$ such that:

- Given a received word $\mathbf{y}$, $\mathcal{A}$ computes the list of codewords $\mathbf{c}_1, \ldots, \mathbf{c}_m \in C$ satisfying $d_H(\mathbf{c}_i, \mathbf{y}) \leq \tau$.

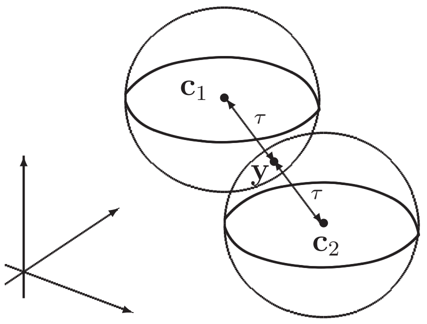


Figure 1.2: When $d(C) \geq 2\tau$ it can happen that the Hamming balls of radius τ around two codewords $\mathbf{c}_1$ and $\mathbf{c}_2$, both contain the received word $\mathbf{y}$.

- The running time of $\mathcal{A}$ is polynomial in n.

The fraction τ/n is called the (relative) *error-correcting radius* or simply error-radius of $\mathcal{A}$. We will express the fact that $\mathcal{A}$ is an τ–list decoder for C by saying that it *decodes C up to error-radius τ/n.*

Note that the last item of the definition implicitly requires the list decoder to find polynomially sized lists (measured in n), since the list decoder will not have time to output, and much less to compute, larger lists.

1.1.4 Limits to list decoding

The following theorem bounds how large error-radii are possible for list decoding while still maintaining polynomial list sizes:

Theorem 1.2. *Let C be an error-correcting code of length n over Σ. Let $q = |\Sigma|$ and assume that $0 < \tau/n < 1 - 1/q$. Let $H_q(p)$ denote the q-ary entropy function*

$$H_q(p) = p \log_q(q-1) - p \log_q(p) - (1-p)\log_q(1-p),$$

and assume that, the rate of C is $R = 1 - H_q(\tau/n) + \varepsilon$ for some $\varepsilon > 0$. Then there exist $\mathbf{y} \in \Sigma^n$ and a list of codewords $\mathbf{c}_1, \ldots, \mathbf{c}_m$ such that m is exponential in n, and such that $d_H(\mathbf{c}_i, \mathbf{y}) \leq \tau$, for $1 \leq i \leq m$.

Proof. We only sketch the proof, see [25, p. 24] for details. Let $B_q(\mathbf{y}, \tau)$ denote the Hamming ball of radius τ with center $\mathbf{y}$, $B_q(\mathbf{y}, \tau) = \{\mathbf{x} \in \Sigma^n \mid$

$d_H(\mathbf{x},\mathbf{y}) \le \tau\}$. It holds that

$$\lim_{h\to\infty} \frac{\log_q(|B_q(\mathbf{y},\tau/n\cdot h)|)}{h} = H_q(\tau/n),$$

and hence for sufficiently large n we have $|B_q(\mathbf{y},\tau)| \approx q^{nH_q(\tau/n)}$. Choose $\mathbf{y}$ uniformly random in Σ^n and let Z be the random variable $Z = |B_q(\mathbf{y},\tau) \cap C|$. We can compute the expectation of Z as

$$\begin{aligned} E_{\mathbf{y}}[Z] &= \sum_{\mathbf{c}\in C} \frac{|B_q(\mathbf{c},\tau)|}{q^n} = |C|\frac{|B_q(\mathbf{0},\tau)|}{q^n} \\ &\approx q^{nR+nH_q(\tau/n)-n} = q^{nR+n(1-R+\varepsilon)-n} = q^{\varepsilon n}. \end{aligned}$$

Hence by the probabilistic method, there exist $\mathbf{y}\in\Sigma^n$ such that $|B_q(\mathbf{y},\tau)\cap C| \approx q^{\varepsilon n}$. □

When q tends to infinity, the q-ary entropy function $H_q(p)$ tends to p. Therefore the above theorem shows that for large alphabets, if the error-radius τ is such that $\tau/n = 1-R+\varepsilon$, then there exists a received word, for which a τ–list decoder is required to output an exponential number of codewords. Thus list decoding with such a large error-radius is not possible. One way to phrase this result, which we will be using in the following, is that $1-R$ is the *capacity* of list decoding.

As a result in the other direction the so-called Johnson bound [22], shows that any linear code of minimum distance d can be list decoded with error-radius τ satisfying Equation (1.1). In particular codes attaining the Singleton bound, mentioned above, are list decodable for error-radii satisfying $\tau/n < 1-\sqrt{R}$.

Theorem 1.3. *Let C be an error-correcting code of length n, defined over an alphabet Σ of size q. Let d denote the minimum distance of C. If $\tau \ge d/2$ and*

$$\tau/n < \left(1-\frac{1}{q}\right)\left(1-\sqrt{1-\frac{q}{q-1}\cdot\frac{d}{n}}\right), \tag{1.1}$$

then for any $\mathbf{y}\in\Sigma^n$ there can be at most

$$\frac{\frac{q}{q-1}\cdot\frac{d}{n}}{\left(1-\frac{q}{q-1}\cdot\frac{\tau}{n}\right)^2 - \left(1-\frac{q}{q-1}\cdot\frac{d}{n}\right)},$$

codewords within distance τ from $\mathbf{y}$.

Proof. See [25, p. 20] or for an elegant proof using embeddings into real vector spaces, [31]. □

It is known that the Johnson bound is tight, in the sense that there exist codes with the property that if τ is chosen slightly larger than the bound in Equation (1.1), then there exist a $\mathbf{y} \in \Sigma^n$ which has a super-polynomial number of codewords within distance τ [22]. Furthermore, the bound is known to be tight for binary linear codes [24]. For a version of the Johnson bound which is independent of the alphabet size, see [15].

1.2 Interpolation and list decoding

When it was introduced in [18, 68], list decoding was a mainly theoretical concept, dominated by non-constructive random coding arguments. It took about thirty years before the first effective list-decoder saw light. In [21] Goldreich and Levin gave a list decoder for Hadamard codes, as a by-product of their investigation of so-called hard-core predicates. The dimension of Hadamard codes is logarithmic in the code length, and hence for long codes its rate is close to zero. Thus such codes are not suitable for high rate data transmission, and this limits the information theoretic relevance of the Goldreich–Levin decoder.

1.2.1 List decoding of Reed–Solomon codes

Unlike the Hadamard codes, Reed–Solomon codes [53] can be constructed to have any (rational) rate in the interval $[0, 1]$. These codes are defined as follows:

Definition 1.4 (Reed–Solomon codes)**.** Let $\mathbb{F}_q$ be a finite field, and let n and k be parameters satisfying $0 \leq k \leq n \leq q$. The Reed–Solomon code of length n and dimension k over $\mathbb{F}_q$ is defined by

$$C = \{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f(x) \in \mathbb{F}_q[x], \deg(f) < k\},$$

where $\alpha_1, \ldots, \alpha_n$ are distinct elements of $\mathbb{F}_q$.

The minimum distance of a Reed–Solomon code is $d = n - k + 1$, and thus it meets the Singleton bound. The Welch–Berlekamp decoder [67] can be used to uniquely decode Reed–Solomon codes when the number of errors satisfy $\tau/n < \frac{1}{2}(1 - R)$. In [12] Berlekamp extended the Welch–Berlekamp decoder, and gave an algorithm capable of correcting one more error than is possible with unique decoding. Thus the algorithm in [12] is in fact a list decoder for Reed–Solomon codes. However, since it only corrects one more error than unique decoding (and not a number which scales with the code length) its advantage over the latter is negligible for long codes.

It was Sudan's great insight that the main principles of [12, 67], could be abstracted and interpreted as a bivariate interpolation problem [65]. Using this, he gave an explicit and very elegant list decoder for Reed–Solomon codes, working for error-radii satisfying $\tau/n < 1 - \sqrt{2R}$. This result was a real breakthrough. Not only was it the first effective list decoder for a class of codes with non-vanishing rate, with performance significantly better than unique decoding, but it also set the direction of future research in list decoding. Since Sudan's result was published, list decoders for a number of codes have appeared in the literature, and it can be taken as measure of the importance of Sudan's result, that all these decoders are based on the principles in [65]. Sudan's list decoding algorithm consists of two steps, an *interpolation step* and a *root-finding step*, and we now outline these.

1. **Interpolation step:** Let $\mathbf{y} \in \mathbb{F}_q^n$ be a received word. Compute the non-zero bivariate polynomial $Q(x,T) = \sum_{i=0}^{\ell} q_i(x)T^i \in \mathbb{F}_q[x,T]$ such that $Q(\alpha_i, y_i) = 0$ for $1 \leq i \leq n$, and such that
$$\deg_{k-1}(Q) = \max\{\deg(q_i(x)) + i(k-1) \mid 0 \leq i \leq \ell, q_i \neq 0\},$$
is least possible under these conditions.

2. **Root-finding step:** Compute and output all polynomials $f(x)$ for which $T - f(x)$ is a factor of $Q(x,T)$ and $\deg(f) < k$.

The polynomial computed in step 1 is called an *interpolation polynomial.* Using a linear algebra argument, it can be shown that there exists an interpolation polynomial such that $\deg_{k-1}(Q) \approx \sqrt{2nk}$. If $f(x)$ is a polynomial generating a Reed–Solomon codeword that agrees with $\mathbf{y}$ in at least $n - \tau$ positions, then the univariate polynomial $Q(x, f(x))$ has at least $n - \tau$ roots. Furthermore, its degree is at most $\sqrt{2nk}$, and hence if $\tau/n < 1 - \sqrt{2R}$, then $Q(x, f(x)) = 0$. Therefore $f(x)$ will be among the polynomials found in the root-finding step.

1.2.2 Improved list decoding of Reed–Solomon codes

Sudan's list decoding algorithm only improves on unique decoding when $R < 1/3$. In [30] Guruswami and Sudan managed to overcome this restriction on the rate, by giving an algorithm capable of list decoding Reed–Solomon codes up to error-radius $\tau/n < 1 - \sqrt{R}$. This is better than unique decoding for all rates, see Figure 1.3. They achieved this improvement by introducing a *multiplicity parameter* s and changing the interpolation step outlined above, to:

1. **Interpolation step:** Let $\mathbf{y} \in \mathbb{F}_q^n$ be a received word. Compute the non-zero bivariate polynomial $Q(x,T) = \sum_{i=0}^{\ell} q_i(x)T^i \in \mathbb{F}_q[x,T]$ such that $Q(x,T)$ has a zero of multiplicity at least s in the points (α_i, y_i) for $1 \leq i \leq n$, and such that

$$\deg_{k-1}(Q) = \max\{\deg(q_i(x)) + i(k-1) \mid 0 \leq i \leq \ell, q_i \neq 0\},$$

is least possible under these conditions.

By choosing s and ℓ appropriately, one can show that there exists a polynomial satisfying the requirements in this interpolation step, such that

$$\deg_{k-1}(Q) \approx s\sqrt{nk}.$$

Let $f(x)$ be as before, then the polynomial $Q(x, f(x))$ has at least $s(n-\tau)$ roots (counting multiplicities) and since its degree is at most $s\sqrt{nk}$, it follows that $Q(x, f(x)) = 0$ if $\tau/n < 1 - \sqrt{R}$. Thus the list decoder in [30] shows that for Reed–Solomon codes there exists an explicit list decoder which corrects errors up to the Johnson bound in Theorem 1.3.

Above we have formulated what is often called the *Guruswami–Sudan list decoding algorithm* for Reed–Solomon codes. We will use this decoder repeatedly in the following, and for brevity we will refer to it as the G–S algorithm. In [30] the algorithm was formulated for general algebraic-geometry codes, building on the work of Shokrollahi and Wasserman [62], and in [51] Pellikaan and Wu extended it to general Reed–Muller codes. Thus the G–S list decoding algorithm actually applies to a broad class of algebraic codes. Furthermore, recently the G–S algorithm has been extended to a combinatorial tool (or principle) which has been used to resolve the Kakeya conjecture for finite fields [17]. Together, these facts demonstrate the power and generality of the G–S algorithm.

1.2.3 Approaching capacity

In Theorem 1.2 we saw that it is not possible to list decode an error-correcting code beyond the capacity $1 - R$. On the other hand, the G–S algorithm

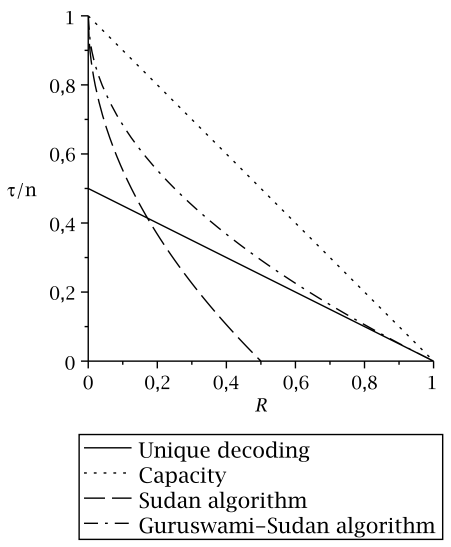


Figure 1.3: The relationship between *rate* R and *error-correcting radius* τ/n for various bounds and list decoders.

promises us that there exist explicit codes, with matching decoders, that can be list decoded up to error-radius $\tau/n < 1 - \sqrt{R}$. Therefore a natural question is: can one find an explicit code that can be list decoded up to an error-radius larger than $1 - \sqrt{R}$, and can this be achieved with an explicit list decoder? In [50] Parvaresh and Vardy gave a variant of Reed–Solomon codes and showed that by extending the interpolation step in the G–S list decoder to $(v+1)$-variate polynomials, these codes can be list decoded up to error-radius

$$\tau/n < 1 - (vR)^{\frac{v}{v+1}}. \tag{1.2}$$

Here $v \geq 1$ is a parameter that can be chosen freely, and the case $v = 1$ corresponds to ordinary Reed–Solomon codes. For rates $R < 1/16$ this result improves upon the Guruswami–Sudan list decoding radius $1 - \sqrt{R}$. It is the factor $v^{\frac{v}{v+1}}$ appearing in Equation (1.2) that is the hindrance for the list decoder in [50] to be useful for high rate codes. In [27–29] Guruswami and Rudra added *folding* to the codes considered by Parvaresh and Vardy, in order to get rid of the $v^{\frac{v}{v+1}}$ factor. Furthermore, they gave an explicit list decoder capable of list decoding the resulting *folded Reed–Solomon codes*, up to error-radius

$$\tau/n < 1 - R^{\frac{v}{v+1}},$$

for sufficiently large alphabets. Letting v tend to infinity, Guruswami and Rudra's result shows that the bound in Theorem 1.2 is tight: It is possible to construct an explicit code with an explicit list decoder capable of list decoding up to error-radii arbitrarily close to $1 - R$. On the other hand *no* code can be list decoded with an error-radius which is just slightly larger than $1 - R$.

1.3 This thesis

As demonstrated in the previous section all known effective list decoding algorithms are based on the principles of the G–S algorithm. In particular, these list decoding algorithms rely, in one way or the other, on solving a multivariate interpolation problem with certain degree constraints. In this thesis we will investigate how efficient list decoding algorithms based on the principles of the G–S algorithm can be implemented for a class of algebraic-geometry codes and for folded Reed–Solomon codes. In the investigations of list decoders for algebraic-geometry codes we will focus on the interpolation step, since in practise this is often the most time consuming part of the decoders. For folded Reed–Solomon codes we will investigate both the interpolation step and the root-finding step in detail.

The thesis is organized as follows:

- **Chapter 2:** In this chapter a general interpolation problem, encompassing all interpolation problems encountered in later chapters, is set up. Furthermore, the interpolation problem is reformulated in terms of so-called key-equations. These key-equations will provide a convenient interface to the interpolation problem, which will be used in later chapters.

- **Chapter 3:** This chapter describes a general algorithm for computing "short" bases of modules over a univariate polynomial ring, when the shortness is measured with respect to certain weighted degrees. This algorithm will be the workhorse in the interpolation algorithm described in the next chapter. The algorithm uses a divide–and–conquer technique, which makes it efficient for modules defined by a "long" basis (with respect to the weighted degree). The chapter provides a detailed analysis of the algorithm's asymptotic performance.

- **Chapter 4:** The key-equations derived in Chapter 2 translate the interpolation problem to a problem of finding "short" vectors in certain modules. In this chapter this fact is exploited to build an interpolation algorithm on top of the short-basis algorithm from Chapter 3. Furthermore, this interpolation algorithm is applied to the interpolation step of the G–S algorithm for a class of algebraic-geometry codes. It is demonstrated that for a number of well-known algebraic-geometry codes, the asymptotic complexity of the resulting algorithm is better than that of previously known algorithms. Furthermore the challenges in making practical implementations of the interpolation algorithm are discussed.

- **Chapter 5:** This chapter contains a detailed exposition of folded Reed–Solomon codes. In particular it is shown that asymptotically these codes have optimal list decoding properties in the sense of Section 1.2.3. Furthermore,
    - It is demonstrated that the interpolation algorithm from Chapter 4 can be applied to efficiently handle the interpolation step in the list decoder for folded Reed–Solomon codes.
    - The algorithm from [27] for the root-finding step of the list decoder, based on viewing codewords as elements of an extension of the code alphabet, is described. Furthermore, the practical performance of this algorithm is assessed.
    - An alternative algorithm for the root-finding step based on Hensel–lifting is developed. No general bounds on the running time of this algorithm are found. However, bounds are derived for special cases or under certain assumptions. Furthermore it is demonstrated that the method works well in practice.
- **Chapter 6:** In this chapter the thesis is summarized and its main results are emphasized. Furthermore, points of interest for future investigations are given.

Chapter 2

Key-equations

In Chapter 1 it was shown that list decoding is closely related to multivariate polynomial interpolation with degree constraints, through the interpolation step in the Guruswami–Sudan list decoding algorithm. A *key-equation* is an equation which quantifies this link: For a decoder which uses the principles of the G–S algorithm, a key-equation gives an exact characterization of all the multivariate polynomials that satisfy the requirements in the decoders interpolation step. The word 'key-equation' was coined by Berlekamp in [10], where he used it for an equation characterizing the decoding problem for binary BCH codes. The equation can be interpreted as a characterization of bivariate polynomials satisfying certain interpolation constraints [38], and for this reason the word key-equation (or key-equations) has, since it first appeared in [10], been used in a variety of decoders that include an interpolation step. The advantage of key-equations is that they have the ability to "compress" any structure or redundancy that may be present in an interpolation problem, and thus solving a set of key-equations is often simpler, than solving the interpolation problem directly.

In this chapter we will formulate a general interpolation problem encompassing all interpolation problems that we will encounter in various list decoders in later chapters. Furthermore, we will derive a set of key-equations characterizing this interpolation problem, and show how these allow us to reformulate the interpolation problem in the language of modules.

A benefit of the generality of our approach is that it allow us to view the different key-equations for Reed–Solomon codes known in the literature, as special cases of our general key-equation set-up. In particular we will see that when interpreted in our language, the different key-equations for Reed–Solomon codes, in fact only differ by simple rewritings.

2.1 Degree constrained multivariate interpolation

In a nutshell we will be interested in multivariate interpolation with multiplicities, where the *interpolation points* are affine rational points on an algebraic curve over a finite field $\mathbb{F}_q$, and the *interpolation values* are tuples of elements in $\mathbb{F}_q$. In the following we will introduce the necessary concepts and notation to make this precise. Furthermore, to make the problem tractable, for the algorithm we propose in Chapter 3 we will need to impose certain restrictions on the curves and interpolation points that we will consider. With notation and restrictions in place, we will end this section with a statement of the general interpolation problem which we will be using in later chapters (see Problem 2.8).

2.1.1 Constraints on curves and interpolation points

An algebraic curve $\mathcal{C}$ over a finite field $\mathbb{F}_q$ can be equivalently described by its function field $\mathbb{F}_q(\mathcal{C})$, and in the following we will be using the function field point of view. See [63] for a comprehensive introduction to the theory of function fields. In all of the following we will restrict ourselves to interpolation problems where the interpolation points are rational points on a so-called simple C_{ab} curve. Thus in the following $\mathcal{C}$ will always denote such a curve. To avoid breaking the flow of this section, we postpone a detailed description of simple C_{ab} curves and their properties to Section 2.3. Below we summarize the results of Section 2.3 needed for the developments in the remainder of this section:

- We will denote an absolutely irreducible polynomial defining the function field extension $\mathbb{F}_q(\mathcal{C})/\mathbb{F}_q(x_1)$ by $F(X_1, X_2)$, so that $\mathbb{F}_q(\mathcal{C}) = \mathbb{F}_q(x_1, x_2)$, where $F(x_1, x_2) = 0$.
- On a simple C_{ab} curve x_1 and x_2 have a unique pole P_∞. This claim will be proved in Proposition 2.16.
- We will denote the degree of the function field extension $\mathbb{F}_q(\mathcal{C})/\mathbb{F}_q(x_1)$ by γ, i.e. $\gamma = \deg_{X_2}(F)$.
- We will denote the the ring of functions that only have poles at P_∞ by,

$$\mathcal{R} = \bigcup_{m=0}^{\infty} \mathcal{L}(mP_\infty). \tag{2.1}$$

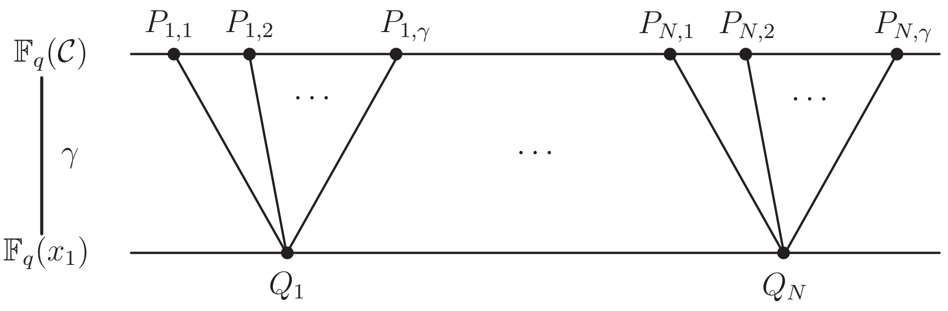


Figure 2.1: The places $P_1, \ldots, P_n$ can be grouped so that the i-th group $P_{i,1}, \ldots, P_{i,\gamma}$ all lie above the same place Q_i.

We also need to impose restrictions on the interpolation points. In the following we will only consider interpolation problems for which the interpolation points satisfy the following assumption:

Assumption 2.1. We assume that $P_1, \ldots, P_n$ are rational places of $\mathbb{F}_q(\mathcal{C})$ different from P_∞, for which it holds that:

- $P_1, \ldots, P_n$ can be divided into $N = n/\gamma$ groups of size γ, such that the places in the i-th group $P_{i,1}, \ldots, P_{i,\gamma}$ all lie above the same place Q_i in $\mathbb{F}_q(x_1)$. See Figure 2.1.

We will denote the set of places $\{Q_1, \ldots, Q_N\}$ by S.

The grouping assumption above is equivalent to the statement that for $1 \leq i \leq N$ the place Q_i splits completely in the extension $\mathbb{F}_q(\mathcal{C})/\mathbb{F}_q(x_1)$. Therefore, since

$$v_{P_{i,j}}\left(x_1 - x_1(Q_i)\right) = e(P_{i,j} \mid Q_i) \cdot v_{Q_i}\left(x_1 - x_1(Q_i)\right).$$

the assumption can be stated equivalently by saying that the function $E_i = x_1 - x_1(Q_i)$ must have divisor

$$(E_i) = \sum_{j=1}^{\gamma} P_{i,j} - \gamma P_\infty. \tag{2.2}$$

In particular, the assumption implies that the function $E = \prod_{i=1}^{N} E_i$ has divisor

$$(E) = \sum_{i=1}^{n} P_i - nP_\infty. \tag{2.3}$$

In the univariate polynomial ring $\mathbb{F}_q[x_1]$ a polynomial U has $\alpha \in \mathbb{F}_q$ as a root of multiplicity at least s if and only if $(x_1 - \alpha)^s$ divides U. The reason for making Assumption 2.1 comes from the need of a similar fact to hold for the ring $\mathcal{R}$. The following proposition shows that we have achieved this.

Proposition 2.2. *For $H \in \mathbb{F}_q(\mathcal{C})$, $H \neq 0$ it holds that*

$$\mathcal{L}\left(-(H) + \infty P_\infty\right) = H\mathcal{R}. \tag{2.4}$$

Proof. We verify that the left hand side of (2.4) is contained in the right hand side, and vice versa. First let $U \in \mathcal{L}\left(-(H) + \infty P_\infty\right)$, then there exist $h \geq 0$ such that $(U) \geq (H) - hP_\infty$. Therefore $(U/H) \geq -hP_\infty$ which means that U/H is an element in $\mathcal{R}$. This proves one inclusion. On the other hand, let $U \in \mathcal{R}$, then there exist $h \geq 0$ such that $(U) \geq -hP_\infty$ which means that $(HU) \geq (H) - hP_\infty$. Therefore HU lies in $\mathcal{L}\left(-(H) + \infty P_\infty\right)$ as desired. □

Equation (2.3) and the above proposition show that $U \in \mathcal{R}$ has a zero of multiplicity at least s in *all* of the places $P_1, \ldots, P_n$ if and only if E^s divides U in $\mathcal{R}$. This is analogous to the situation in univariate polynomial rings outlined above.

2.1.2 Zeroes of multivariate polynomials

In the following we will be working with multivariate polynomials in v indeterminates with coefficients in $\mathcal{R}$. We use vector notation for multivariate polynomials, so that $\mathcal{R}[\mathbf{T}] = \mathcal{R}[T_1, \ldots, T_v]$ and for $\mathbf{R} = (R_1, \ldots, R_v) \in \mathbb{F}_q[\mathbf{T}]^v$ and $\mathbf{i} = (i_1, \ldots, i_v)$,

$$\mathbf{R}^{\mathbf{i}} = \prod_{h=1}^{v} R_h^{i_h}.$$

To increase readability we drop any dependence on v from the notation, and instead let it be fixed and understood throughout. In the following we will repeatedly need to index simplex–like regions of $\mathbb{N}_0^v$ and thus we introduce the following index-set:

$$\Delta_\ell = \left\{\mathbf{b} \in \mathbb{N}_0^v \mid b_1 + \cdots + b_v \leq \ell\right\}.$$

We will refer to the size of this set by

$$m_\ell = |\Delta_\ell| = \binom{\ell + v}{v}. \tag{2.5}$$

We will also use the convention that

$$\binom{\mathbf{j}}{\mathbf{i}} = \prod_{h=1}^{v} \binom{j_h}{i_h}.$$

With this vector notation for binomial coefficients, the so-called Hasse–derivative of a polynomial can be defined as follows.

Definition 2.3 (Hasse–derivative)**.** For $\mathbf{b} \in \mathbb{N}_0^v$ the $\mathbf{b}$-th Hasse–derivative is the $\mathcal{R}$–linear function $\mathcal{H}_{\mathbf{T}}^{(\mathbf{b})} : \mathcal{R}[\mathbf{T}] \to \mathcal{R}[\mathbf{T}]$ satisfying

$$\mathcal{H}_{\mathbf{T}}^{(\mathbf{b})}\left(\mathbf{T}^{\mathbf{i}}\right) = \binom{\mathbf{i}}{\mathbf{b}} \mathbf{T}^{\mathbf{i}-\mathbf{b}}.$$

For details on Hasse–derivatives see [44, p. 303]. With the aid of Hasse–derivatives we can now make precise what it means for a polynomial $Q(\mathbf{T}) \in \mathcal{R}[\mathbf{T}]$ to have a multiple zero.

Definition 2.4 (Multiplicity of zero)**.** Let $Q(\mathbf{T}) \in \mathcal{R}[\mathbf{T}]$. For a place P of $\mathbb{F}_q(\mathcal{C})$ different from P_∞ and a tuple $\mathbf{y} \in \mathbb{F}_q^v$, the polynomial $Q(\mathbf{T})$ is said to have a zero of multiplicity s in $(P, \mathbf{y})$, if s is the largest integer such that for all $\mathbf{b} \in \Delta_{s-1}$ it holds that

$$v_P\left(\mathcal{H}_{\mathbf{T}}^{(\mathbf{b})}(Q)(\mathbf{y})\right) \geq s - \sum_{h=1}^{v} b_h.$$

2.1.3 Interpolation polynomials

With notation in place, we can now make precise what we will understand by an interpolation polynomial.

Definition 2.5 (Interpolation polynomial)**.** Let $\mathcal{C}$ be a simple C_{ab} curve. Let there be given

- Interpolation points: Distinct places $\mathbf{P} = (P_1, \ldots, P_n)$ of $\mathbb{F}_q(\mathcal{C})$, satisfying Assumption 2.1.
- Interpolation values: Tuples $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$, where each $\mathbf{y}_i$ is an element of $\mathbb{F}_q^v$.
- Multiplicity parameter: Integer $s \geq 1$.
- Degree bound: Integer $\ell \geq s$.

A polynomial $Q(\mathbf{T}) \in \mathcal{R}[\mathbf{T}]$ is said to be an interpolation polynomial for $(\mathbf{P}, \mathbf{Y}, s, \ell)$, if it holds that:

(i) Q is non-zero.

(ii) The total degree of Q in the indeterminates $\mathbf{T}$ is at most ℓ.

(iii) Q has a zero of multiplicity at least s in the points $(P_1, \mathbf{y}_1), \ldots, (P_n, \mathbf{y}_n)$.

Usually $(\mathbf{P}, \mathbf{Y}, s, \ell)$ is clear from the context, and in this case we will simply say that $Q(\mathbf{T})$ is an interpolation polynomial. In Theorem 2.12 in the next section we shall see that we can characterize interpolation polynomials completely, as polynomials $Q(\mathbf{T}) = \sum_{\mathbf{j}} q_{\mathbf{j}} \mathbf{T}^{\mathbf{j}}$ for which the coefficient vector $\mathbf{q}$ is an element in the column span of an explicit matrix.

2.1.4 Weighted degree constraints

For our applications in list decoding, we will need to impose certain degree constraints on the interpolation polynomials. To formulate these constraints we need the following notion of weighted degree.

Definition 2.6 (Weighted degree)**.** Let $\mathbf{w} \in \mathbb{Z}^v$ be a vector of *weights* and let $\mathbf{q} = (q_{\mathbf{j}})_{\mathbf{j} \in \Delta_\ell}$ be a vector in $\mathcal{R}^{m_\ell}$. Then we define the following weighted degree of $\mathbf{q}$,

$$\deg_{\mathbf{w}}(\mathbf{q}) = \begin{cases} \max_{\mathbf{j} \in \Delta_\ell, q_{\mathbf{j}} \neq 0} \left\{ -v_{P_\infty}(q_{\mathbf{j}}) + \mathbf{w} \cdot \mathbf{j} \right\} & \text{if } \mathbf{q} \neq \mathbf{0}, \\ -\infty & \text{if } \mathbf{q} = \mathbf{0}, \end{cases}$$

where $\mathbf{w} \cdot \mathbf{j}$ denotes the usual inner product of $\mathbf{w}$ and $\mathbf{j}$. Furthermore, for a polynomial $Q(\mathbf{T}) = \sum_{\mathbf{j} \in \Delta_\ell} q_{\mathbf{j}} \mathbf{T}^{\mathbf{j}} \in \mathcal{R}[\mathbf{T}]$ we define its weighted degree to be the weighted degree of the coefficient vector, i.e. $\deg_{\mathbf{w}}(\mathbf{q})$.

We emphasize the following special case of the above definition: An element A of $\mathcal{R}$ can be considered a constant polynomial in $\mathcal{R}[\mathbf{T}]$, and therefore the weighted degree of A is

$$\deg_{\mathbf{w}}(A) = -v_{P_\infty}(A).$$

We can now define the special type of interpolation polynomials, that we will be interested in, namely interpolation polynomials satisfying certain weighted degree constraints.

Definition 2.7 (Valid interpolation polynomial)**.** Let $(\mathbf{P}, \mathbf{Y}, s, \ell)$ be as in Definition 2.5, and let $\mathbf{w} \in \mathbb{Z}^v$ be a *weight vector*. Let there be given a *weighted degree bound* $\Delta \geq 0$, then a polynomial $Q(\mathbf{T})$ is said to be a *valid* interpolation polynomial for $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$ if it holds that

(i) Q is an interpolation polynomial for $(\mathbf{P}, \mathbf{Y}, s, \ell)$.

(ii) $\deg_{\mathbf{w}}(Q(\mathbf{T})) < \Delta$.

As before, the quantity $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$ is usually clear from the context, in which case we will simply say that $Q(\mathbf{T})$ is a valid interpolation polynomial.

2.1.5 The interpolation problem

We are now ready to formulate our central interpolation problem. As mentioned earlier, all interpolation problems which we will encounter in the following are special cases of this general problem.

Problem 2.8 (Interpolation problem)**.** Let $\mathcal{C}$ be a simple C_{ab} curve, and let $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$ be parameters:

- Interpolation points: Distinct places $\mathbf{P} = (P_1, \ldots, P_n)$ of $\mathbb{F}_q(\mathcal{C})$, satisfying Assumption 2.1
- Interpolation values: Tuples $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$, where each $\mathbf{y}_i$ is an element of $\mathbb{F}_q^v$.
- Multiplicity parameter: Integer $s \geq 1$.
- Degree bound: Integer $\ell \geq s$.
- Weight vector: Vector $\mathbf{w} \in \mathbb{Z}^v$.
- Weighted degree bound: Integer Δ.

We define the *interpolation problem* for these parameters, to be the problem of finding a valid interpolation polynomial for $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$, if it exists.

Remark 2.9. Ideally we would like to be able to compute a valid interpolation polynomial (if it exists) for given $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$, without any assumptions on the curve and the interpolation points. While effective algorithms achieving this exist [56, 57], the approach we will focus on in the following can not handle completely general curves and interpolation points.

As we shall see in Chapter 4, the algorithm we give for computing valid interpolation polynomials will be faster than those in [56, 57]. Thus, confining ourselves to simple C_{ab} curves and special interpolation points may be viewed as a sacrifice of generality at benefit of speed.

2.2 Reformulation of the interpolation problem

In this section we will derive a set of key-equations characterizing the interpolation problem in Problem 2.8. We will show that the interpolation problem can be equivalently stated as a problem of finding a "short" vector in a module over $\mathcal{R}$. Later we will exploit this to give an efficient algorithm for solving Problem 2.8.

2.2.1 A family of matrices

In our derivation of key-equations for the interpolation problem, the following family of matrices will be central.

Definition 2.10. Let $\ell \geq 0$ be an integer. Define $\mathbf{A}_\ell : \mathcal{R}^v \to \mathrm{Mat}_{m_\ell \times m_\ell}(\mathcal{R})$ to be the function given by

$$[\mathbf{A}_\ell(\mathbf{T})]_{\mathbf{i},\mathbf{j}} = \mathcal{H}_{\mathbf{T}}^{(\mathbf{j})}\left(\mathbf{T}^{\mathbf{i}}\right) = \binom{\mathbf{j}}{\mathbf{i}} \mathbf{T}^{\mathbf{j}-\mathbf{i}}, \tag{2.6}$$

where the row and columns indices $\mathbf{i}$ and $\mathbf{j}$ both run in Δ_ℓ. Furthermore, for integers $s \leq \ell$, define $\mathbf{D}_{s,\ell} : \mathcal{R} \to \mathrm{Mat}_{m_\ell \times m_\ell}(\mathcal{R})$ to be the function given by

$$[\mathbf{D}_{s,\ell}(T)]_{\mathbf{i},\mathbf{j}} = \begin{cases} T^{\max(s - \sum_{h=1}^{v} i_h, 0)} & \text{for } \mathbf{i} = \mathbf{j} \\ 0 & \text{otherwise,} \end{cases} \tag{2.7}$$

where again the indices $\mathbf{i}$ and $\mathbf{j}$ both run in Δ_ℓ. Note that $\mathbf{D}_{s,\ell}(T)$ is a diagonal matrix and that the number of rows and columns of $\mathbf{A}_\ell(\mathbf{T})$ and of $\mathbf{D}_{s,\ell}(T)$ is m_ℓ.

The matrices in the above definition satisfy a number of relations, which we establish in the following proposition.

Proposition 2.11. *Let $\mathbf{S} = (S_1, \ldots, S_v)$ and $\mathbf{T} = (T_1, \ldots, T_v)$ be vectors in $\mathcal{R}^v$, and let S and T be elements in $\mathcal{R}$. It holds that:*

(i) $\mathbf{A}_\ell(\mathbf{S})\mathbf{A}_\ell(\mathbf{T}) = \mathbf{A}_\ell(\mathbf{S}+\mathbf{T})$.

(ii) $\mathbf{A}_\ell(\mathbf{T})^{-1} = \mathbf{A}_\ell(-\mathbf{T})$.

(iii) $\mathbf{D}_{s,\ell}(S)\mathbf{D}_{s,\ell}(T) = \mathbf{D}_{s,\ell}(ST)$.

Proof. We prove each part of the proposition separately.

(i) By (2.6) it holds that

$$\begin{aligned}
[\mathbf{A}_\ell(\mathbf{S})\mathbf{A}_\ell(\mathbf{T})]_{\mathbf{i},\mathbf{j}} &= \sum_{\mathbf{k}\in\Delta_\ell} [\mathbf{A}_\ell(\mathbf{S})]_{\mathbf{i},\mathbf{k}} [\mathbf{A}_\ell(\mathbf{T})]_{\mathbf{k},\mathbf{j}} \\
&= \sum_{\mathbf{k}\in\Delta_\ell} \binom{\mathbf{k}}{\mathbf{i}}\binom{\mathbf{j}}{\mathbf{k}} \mathbf{S}^{\mathbf{k}-\mathbf{i}}\mathbf{T}^{\mathbf{j}-\mathbf{k}} \\
&= \sum_{\mathbf{k}\in\Delta_\ell} \binom{\mathbf{j}}{\mathbf{i}}\binom{\mathbf{j}-\mathbf{i}}{\mathbf{j}-\mathbf{k}} \mathbf{S}^{\mathbf{k}-\mathbf{i}}\mathbf{T}^{\mathbf{j}-\mathbf{k}} \\
&= \binom{\mathbf{j}}{\mathbf{i}} \sum_{\mathbf{k}\in\mathbf{j}-\Delta_\ell} \binom{\mathbf{j}-\mathbf{i}}{\mathbf{k}} \mathbf{S}^{\mathbf{j}-\mathbf{i}-\mathbf{k}}\mathbf{T}^{\mathbf{k}} \\
&= \binom{\mathbf{j}}{\mathbf{i}} (\mathbf{S}+\mathbf{T})^{\mathbf{j}-\mathbf{i}} \\
&= [\mathbf{A}_\ell(\mathbf{S}+\mathbf{T})]_{\mathbf{i},\mathbf{j}}\,.
\end{aligned}$$

(ii) By *(i)* we have that $\mathbf{A}_\ell(\mathbf{T})\mathbf{A}_\ell(-\mathbf{T}) = \mathbf{A}_\ell(\mathbf{0}) = \mathbf{I}_{m_\ell}$.

(iii) We have

$$[\mathbf{D}_{s,\ell}(S)\mathbf{D}_{s,\ell}(T)]_{\mathbf{i},\mathbf{j}} = \sum_{\mathbf{k}\in\Delta_\ell} [\mathbf{D}_{s,\ell}(S)]_{\mathbf{i},\mathbf{k}} [\mathbf{D}_{s,\ell}(T)]_{\mathbf{k},\mathbf{j}}\,.$$

By (2.7) the last expression is equal to

$$S^{\max(s-\sum_{h=1}^{v} i_h,0)}T^{\max(s-\sum_{h=1}^{v} i_h,0)} = (ST)^{\max(s-\sum_{h=1}^{v} i_h,0)},$$

if $\mathbf{i}=\mathbf{j}$ and zero otherwise.

□

2.2.2 Key-equations for the interpolation problem

We are now ready to derive the desired set of key-equations characterizing the interpolation problem in Problem 2.8.

Theorem 2.12. *Let the curve $\mathcal{C}$ and the interpolation parameters $(\mathbf{P}, \mathbf{Y}, s, \ell)$ be as in Problem 2.8. Let $\mathbf{R} = (R_1, \dots, R_v) \in \mathcal{R}^v$ be a vector of Lagrangian interpolation polynomials satisfying $R_j(P_i) = y_{ij}$ for $1 \leq i \leq n$ and $1 \leq j \leq v$. Then for a polynomial*

$$Q(\mathbf{T}) = \sum_{\mathbf{j} \in \Delta_\ell} q_{\mathbf{j}} \mathbf{T}^{\mathbf{j}} \in \mathcal{R}[\mathbf{T}]$$

with coefficient vector $\mathbf{q} = (q_{\mathbf{j}})$, the following are equivalent:

(i) $Q(\mathbf{T})$ is an interpolation polynomial for $(\mathbf{P}, \mathbf{Y}, s, \ell)$.

(ii) There exists a vector $\mathbf{c} \in \mathcal{R}^{m_\ell}$ such that $\mathbf{q} = \mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)\mathbf{c}$.

Proof. In the following $\mathbf{b} = (b_1, \dots, b_v)$ and $\mathbf{j} = (j_1, \dots, j_v)$ will denote vectors in Δ_ℓ. For a divisor A of $\mathbb{F}_q(\mathcal{C})$, we define the following Cartesian product of $\mathcal{R}$–modules

$$\mathcal{L}_A = \prod_{\mathbf{b} \in \Delta_\ell} \mathcal{L}\left(-\max(0, s - \textstyle\sum_{h=1}^{v} b_h)A + \infty P_\infty\right).$$

Furthermore, we let D denote the divisor $D = P_1 + \dots + P_n$. By the Taylor–expansion formula for the Hasse–derivative, it holds that

$$Q(\mathbf{T}) = \sum_{\mathbf{b} \in \Delta_\ell} \mathcal{H}_{\mathbf{T}}^{(\mathbf{b})}(Q)(\mathbf{y}_i)(\mathbf{T} - \mathbf{y}_i)^{\mathbf{b}}. \tag{2.8}$$

From this equation it follows that $Q(\mathbf{T})$ has a zero in $(P_i, \mathbf{y}_i)$ of multiplicity at least s if and only if

$$\mathcal{H}_{\mathbf{T}}^{(\mathbf{b})}(Q)(\mathbf{y}_i) \in \mathcal{L}\left(-(s - \textstyle\sum_{h=1}^{v} b_h)P_i + \infty P_\infty\right), \tag{2.9}$$

for all $\mathbf{b}$ such that $\sum_{h=1}^{v} b_h < s$. Note that by definition of $\mathbf{A}_\ell$,

$$[\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q}]_{\mathbf{b}} = \sum_{\mathbf{k} \in \Delta_\ell} [\mathbf{A}_\ell(\mathbf{y}_i)]_{\mathbf{b},\mathbf{k}} q_{\mathbf{k}} = \sum_{\mathbf{k} \in \Delta_\ell} \binom{\mathbf{k}}{\mathbf{b}} \mathbf{y}_i^{\mathbf{k}-\mathbf{b}} q_{\mathbf{k}} = \mathcal{H}_{\mathbf{T}}^{(\mathbf{b})}(Q)(\mathbf{y}_i),$$

and hence the $\mathbf{b}$-th entry of $\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q}$ is equal to the left hand side of (2.9). Therefore item (i) is equivalent to *(iii)* For $1 \leq i \leq n$ it holds that $\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q} \in \mathcal{L}_{P_i}$.

Since $R_j(P_i) = y_{ij}$ we can write $R_j = y_{ij} + S_j$, where $S_j \in \mathcal{L}(-P_i + \infty P_\infty)$. Using vector notation we may write this as $\mathbf{R} = \mathbf{y}_i + \mathbf{S}$. We claim that *(iii)* is equivalent to

(iv) For $1 \leq i \leq n$ it holds that $\mathbf{A}_\ell(\mathbf{R})\mathbf{q} \in \mathcal{L}_{P_i}$.

To see this, fix i and assume that $\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q} \in \mathcal{L}_{P_i}$. From Proposition 2.11 we have that

$$\mathbf{A}_\ell(\mathbf{R})\mathbf{q} = \mathbf{A}_\ell(\mathbf{S})\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q}.$$

The $(\mathbf{b},\mathbf{j})$-th entry in $\mathbf{A}_\ell(\mathbf{S})$ is $\binom{\mathbf{j}}{\mathbf{b}}\mathbf{S}^{\mathbf{j}-\mathbf{b}}$, and hence

$$[\mathbf{A}_\ell(\mathbf{S})]_{\mathbf{b},\mathbf{j}} \in \mathcal{L}\left(-\textstyle\sum_{h=1}^{v} \max(0, j_h - b_h)P_i + \infty P_\infty\right).$$

Furthermore, by assumption

$$[\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q}]_{\mathbf{j}} \in \mathcal{L}\left(-\max(0, s - \textstyle\sum_{h=1}^{v} j_h)P_i + \infty P_\infty\right).$$

Therefore, since

$$[\mathbf{A}_\ell(\mathbf{R})\mathbf{q}]_{\mathbf{b}} = [\mathbf{A}_\ell(\mathbf{S})\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q}]_{\mathbf{b}} = \sum_{\mathbf{j}\in\Delta_\ell} [\mathbf{A}_\ell(\mathbf{S})]_{\mathbf{b},\mathbf{j}}[\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q}]_{\mathbf{j}}$$

and since

$$\begin{aligned}&\textstyle\sum_{h=1}^{v} \max(0, j_h - b_h) + \max(0, s - \sum_{h=1}^{v} j_h) \geq \\ &\quad \max(0, \textstyle\sum_{h=1}^{v}(j_h - b_h)) + \max(0, s - \sum_{h=1}^{v} j_h) \geq \max(0, s - \sum_{h=1}^{v} b_h),\end{aligned}$$

it follows that $\mathbf{A}_\ell(\mathbf{R})\mathbf{q} \in \mathcal{L}_{P_i}$. Thus *(iii)* implies *(iv)*. To prove the reverse implications, fix i and assume that $\mathbf{A}_\ell(\mathbf{R})\mathbf{q} \in \mathcal{L}_{P_i}$. By Proposition 2.11 we have

$$\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q} = \mathbf{A}_\ell(-\mathbf{S})\mathbf{A}_\ell(\mathbf{R})\mathbf{q},$$

and using this expression we can argue exactly as above to get that $\mathbf{A}_\ell(\mathbf{y}_i)\mathbf{q} \in \mathcal{L}_{P_i}$, as desired. Since the vector $\mathbf{A}_\ell(\mathbf{R})\mathbf{q}$ does not depend on the index i we get that *(iv)* is equivalent to

(v) It holds that $\mathbf{A}_\ell(\mathbf{R})\mathbf{q} \in \bigcap_{i=1}^{n} \mathcal{L}_{P_i} = \mathcal{L}_D$.

From Equation (2.3) we know that $(E) = D - nP_\infty$, and therefore Proposition 2.2 gives that

$$\mathcal{L}\left(-\max(0, s - \textstyle\sum_{h=1}^{v} b_h)D + \infty P_\infty\right) = E^{\max(0,s-\sum_h b_h)}\mathcal{R}.$$

Therefore, by definition of the matrix $\mathbf{D}_{s,\ell}$, we have that (v) is equivalent to the existence of a vector $\mathbf{c} \in \mathcal{R}^{m_\ell}$ such that

$$\mathbf{A}_\ell(\mathbf{R})\mathbf{q} = \mathbf{D}_{s,\ell}(E)\mathbf{c}.$$

Finally by Proposition 2.11 this is equivalent to item (ii), and thus the above chain of equivalences prove the theorem. □

Remark 2.13. In the proof of Theorem 2.12 above, the only assumption on the interpolation points we used, is Equation (2.3). Thus for the developments in this section, we could have taken the existence of a function E satisfying Equation (2.3) as our only assumption. However, to develop an efficient algorithm for solving the interpolation problem we will need the stronger properties in Assumption 2.1. Thus in order to avoid formulating an interpolation problem that we can not solve efficiently, we have chosen to state all assumptions already in this section.

2.3 Simple C_{ab} curves

In this section we introduce a class of curves which we call *simple C_{ab} curves*.

2.3.1 Definition of simple C_{ab} curves

As in Section 2.1 we will work with curves in the language of function fields. All curves will be defined over a finite field $\mathbb{F}_q$. If an absolutely irreducible curve $\mathcal{C}$ is defined by an absolutely irreducible bivariate polynomial $F(X_1, X_2) \in \mathbb{F}_q[X_1, X_2]$, then the function field of $\mathcal{C}$ is $\mathbb{F}_q(\mathcal{C}) = \mathbb{F}_q(x_1, x_2)$, where x_1 and x_2 satisfy the relation

$$F(x_1, x_2) = 0, \tag{2.10}$$

and x_1 is a transcendental element over $\mathbb{F}_q$. In the following we will let δ denote $\deg_{X_1}(F)$ and γ will denote $\deg_{X_2}(F)$. We now define the class of simple C_{ab} curves. The curves are special cases of the so-called C_{ab} curves studied in [47], but with extra regularity conditions. One of the conditions is that the curves should be simple (i.e. non-singular), and we thus call them simple C_{ab} curves. It should be mentioned that the curves are also special cases of the so-called *Type I* curves from [19].

Definition 2.14 (Simple C_{ab} curve)**.** A curve $\mathcal{C}$ is a simple C_{ab} curve if it is defined by a polynomial $F(X_1, X_2)$ satisfying:

1. Any monomial $X_1^i X_2^j$ in the support of F satisfy $\gamma i + \delta j \leq \gamma\delta$,
2. $\gcd(\gamma, \delta) = 1$,
3. the ideal $\langle F, \frac{\partial F}{\partial X_1}, \frac{\partial F}{\partial X_2} \rangle$ is equal to the unit ideal of $\mathbb{F}_q[X_1, X_2]$.

Remark 2.15. The name "C_{ab} curves" is perhaps a little unfortunate, as it refers to a specific notation used when defining the curves. In [47] a C_{ab} curve is defined by a polynomial F with $a = \deg_{X_1}(F)$ and $b = \deg_{X_2}(F)$. Thus to follow this terminology we should actually have called the above curves *simple $C_{\delta\gamma}$ curves.* However, since the name "C_{ab} curve" has been widely adopted (as a name, not a notation) we have avoided this.

2.3.2 Properties of simple C_{ab} curves

We now derive a number of properties held by simple C_{ab} curves. In particular we will verify that the curves satisfy the claims made in Section 2.1.1. In the following we will say that a place P of a function field $\mathbb{F}_q(\mathcal{C})$ of a simple C_{ab} curve, is a *place at infinity* if either $v_P(x_1) < 0$ or $v_P(x_2) < 0$.

Proposition 2.16. *Let $\mathcal{C}$ be a simple C_{ab} curve. Then $\mathbb{F}_q(\mathcal{C}) = \mathbb{F}_q(x_1, x_2)$ has exactly one place P_∞ at infinity. Furthermore,*

$$v_{P_\infty}(x_1) = -\gamma, \quad v_{P_\infty}(x_2) = -\delta. \tag{2.11}$$

In particular x_1 and x_2 can only have poles at this place.

Proof. Let P_∞ denote some place of $\mathbb{F}_q(\mathcal{C})$ lying at infinity, i.e. a place such that either $v_{P_\infty}(x_1) < 0$ or $v_{P_\infty}(x_2) < 0$. We will prove that there is only one such place. By definition of simple C_{ab} curves, the Newton polygon of F is as shown in Figure 2.2: It has exactly two points on the line through $(\delta, 0)$ and $(0, \gamma)$, and all other points lie strictly below this line. The valuation at P_∞ of a single monomial $x_1^i x_2^j$ in the support of F is

$$\varphi(i, j) = i v_{P_\infty}(x_1) + j v_{P_\infty}(x_2).$$

We can consider φ as a function mapping from the Newton polygon of F to $\mathbb{Z}$, and since the Newton polygon is a bounded region we know that φ has a minimum and that its minimal value is finite.

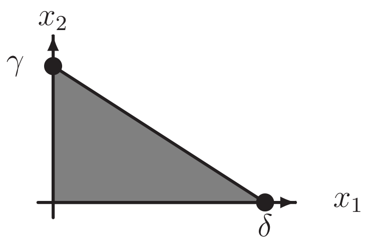


Figure 2.2: The Newton polygon of simple C_{ab} curve.

By symmetry, we may assume that $v_{P_\infty}(x_1) < 0$. If we furthermore assume that $v_{P_\infty}(x_2) \geq 0$, then the graph of φ (when considered as a function from $\mathbb{R}^2$) will have negative slope in the x_1 direction, and non-negative slope in the x_2 direction. Therefore φ has unique minimum, namely $(\delta, 0)$. But by the Strict Triangle Inequality [63, I.1.10] such a unique minimum implies that

$$v_{P_\infty}(F(x_1, x_2)) = \min_{(i,j)} \{\varphi(i, j)\},$$

and since $v_{P_\infty}(F(x_1, x_2)) = v_{P_\infty}(0) = \infty$, this is a contradiction. Therefore $v_{P_\infty}(x_2) < 0$. This means that the graph of φ has negative slope in both the x_1 and the x_2 directions, and hence the minimum of φ must lie on the upper edge of the Newton polygon, i.e. on the line through $(\delta, 0)$ and $(0, \gamma)$. We know that only two points of the Newton polygon lie on this line. Furthermore, arguing as before, we know that the minimum of φ can not be unique, and therefore both of the points $(\delta, 0)$ and $(0, \gamma)$ must be minima of φ. Hence

$$\delta v_{P_\infty}(x_1) = \varphi(\delta, 0) = \varphi(0, \gamma) = \gamma v_{P_\infty}(x_2). \tag{2.12}$$

By [63, I.4.11] we have that $v_{P_\infty}(x_1) \geq -[\mathbb{F}_q(x_1, x_2) : \mathbb{F}_q(x_1)] = -\gamma$ and since $\gcd(\delta, \gamma) = 1$, Equation (2.12) gives that

$$v_{P_\infty}(x_1) = -\gamma, \quad v_{P_\infty}(x_2) = -\delta.$$

Using [63, I.4.11] again, we get that the pole of x_1 in $\mathbb{F}_q(x_1)$ is totally ramified in the extension $\mathbb{F}_q(x_1, x_2)/\mathbb{F}_q(x_1)$ and hence P_∞ is the only place lying above the pole of x_1. Arguing similarly we get that P_∞ is also the only place lying above the pole of x_2 in the extension $\mathbb{F}_q(x_1, x_2)/\mathbb{F}_q(x_2)$. Hence P_∞ is the the only place at infinity of $\mathbb{F}_q(\mathcal{C}) = \mathbb{F}_q(x_1, x_2)$. $\square$

The above proposition shows that simple C_{ab} curves have exactly one place P_∞ at infinity. Thus as in Equation (2.1) we may consider the ring $\mathcal{R} = \bigcup_{m=0}^{\infty} \mathcal{L}(mP_\infty)$. It turns out that for simple C_{ab} curves $\mathcal{R}$ has a very simple description, namely $\mathcal{R} = \mathbb{F}_q[x_1, x_2]$. We derive this result in Proposition 2.19 below, and for this we need the following lemma.

Lemma 2.17. *Let $\widetilde{R}$ denote the ring $\bigcup_{m=0}^{\infty}\mathcal{L}(mP_\infty)$ in the function field $\overline{\mathbb{F}}_q(x_1,x_2)$, where $\overline{\mathbb{F}}_q$ denotes an algebraic closure of $\mathbb{F}_q$. It holds that,*

$$\widetilde{\mathcal{R}} = \overline{\mathbb{F}}_q[x_1,x_2].$$

Proof. In the proof we will use the theory of local rings: for a place P of $\overline{\mathbb{F}}_q(x_1,x_2)$ we denote by $\mathcal{O}_P$ the local ring of P. We denote by $\mathbb{P}_{\overline{\mathbb{F}}_q(x_1,x_2)}$ the set of all places of $\overline{\mathbb{F}}_q(x_1,x_2)$. From [63, III.2.6], it follows that

$$\widetilde{\mathcal{R}} = \bigcap_{P\in\mathbb{P}_{\overline{\mathbb{F}}_q(x_1,x_2)}\setminus\{P_\infty\}} \mathcal{O}_P,$$

is equal to the integral closure of the ring $\overline{\mathbb{F}}_q[x_1]$ in $\overline{\mathbb{F}}_q(x_1,x_2)$, i.e. that

$$\widetilde{\mathcal{R}} = \mathrm{ic}_{\overline{\mathbb{F}}_q(x_1,x_2)}\left(\overline{\mathbb{F}}_q[x_1]\right).$$

Since $F(x_1,x_2)=0$ and F is monic in x_2, we have that x_2 is integral over $\overline{\mathbb{F}}_q[x_1]$. Therefore

$$\overline{\mathbb{F}}_q[x_1] \subseteq \overline{\mathbb{F}}_q[x_1,x_2] \subseteq \widetilde{\mathcal{R}}.$$

Since integral closure preserves inclusion, this implies that

$$\widetilde{\mathcal{R}} = \mathrm{ic}_{\overline{\mathbb{F}}_q(x_1,x_2)}\left(\overline{\mathbb{F}}_q[x_1]\right) \subseteq \mathrm{ic}_{\overline{\mathbb{F}}_q(x_1,x_2)}\left(\overline{\mathbb{F}}_q[x_1,x_2]\right) \subseteq \mathrm{ic}_{\overline{\mathbb{F}}_q(x_1,x_2)}\left(\widetilde{\mathcal{R}}\right) = \widetilde{\mathcal{R}},$$

and hence $\widetilde{\mathcal{R}} = \mathrm{ic}_{\overline{\mathbb{F}}_q(x_1,x_2)}\left(\overline{\mathbb{F}}_q[x_1,x_2]\right)$. Therefore, the proposition will follow if we can show that $\overline{\mathbb{F}}_q[x_1,x_2]$ is integrally closed. Let $\overline{\mathcal{C}}$ denote the affine curve defined by $F(x_1,x_2)=0$ over $\overline{\mathbb{F}}_q$. By [60, p. 126] we have that $\overline{\mathbb{F}}_q[x_1,x_2]$ is integrally closed if $\overline{\mathcal{C}}$ is non-singular. By item 3 in Definition 2.14, this property holds since F defines a simple C_{ab} curve, and thus we conclude that $\overline{\mathbb{F}}_q[x_1,x_2]$ is integrally closed. This proves the lemma. □

In the next lemma we express the genus of a simple C_{ab} curve in terms of γ and δ.

Lemma 2.18. *The genus of a simple C_{ab} curve $\mathcal{C}$, is $\frac{1}{2}(\gamma-1)(\delta-1)$.*

Proof. By Lemma 2.17 we have that $\widetilde{\mathcal{R}} = \overline{\mathbb{F}}_q[x_1,x_2]$. Using the (monic) relation $F(x_1,x_2)=0$ to eliminate powers of x_2 larger than $\gamma-1$, we can thus write any element of $\widetilde{\mathcal{R}}$ on the form

$$A(x_1,x_2) = \sum_{j=0}^{\gamma-1} a_j(x_1)x_2^j, \quad a_j \in \overline{\mathbb{F}}_q[x_1]. \tag{2.13}$$

Let $\bar{P}_\infty$ denote a place lying above P_∞ in the function field extension

$$\overline{\mathbb{F}}_q(x_1, x_2)/\mathbb{F}_q(x_1, x_2),$$

then it follows from [63, III.6.3(a)] that $e(\bar{P}_\infty \mid P_\infty) = 1$. By Proposition 2.16 we therefore have that $v_{\bar{P}_\infty}(x_1) = -\gamma$ and $v_{\bar{P}_\infty}(x_2) = -\delta$. Therefore the valuation at $\bar{P}_\infty$ of one of the summands in Equation (2.13), is of the form

$$v_{\bar{P}_\infty}\left(a_j(x_1)x_2^j\right) = -\deg(a_j(x_1))\gamma - j\delta \equiv -j\delta \bmod \gamma.$$

Therefore, if two summands $a_{j_1}(x_1)x_2^{j_1}$ and $a_{j_2}(x_1)x_2^{j_2}$ have the same valuation at $\bar{P}_\infty$ then $\delta j_1 \equiv \delta j_2 \bmod \gamma$, and since $\gcd(\delta, \gamma) = 1$ and $j_1, j_2 < \gamma$, this implies that $j_1 = j_2$. Thus all summands in (2.13) have distinct valuations at $\bar{P}_\infty$, which by the Strict Triangle Inequality [63, I.1.10] implies that

$$-v_{\bar{P}_\infty}(A(x_1, x_2)) = \max_{j\,:\,a_j \neq 0} \left\{\deg(a_j(x_1))\gamma + j\delta\right\}.$$

This means that the Weierstrass semigroup of $\bar{P}_\infty$ is equal to $\langle \gamma, \delta \rangle$. By [52, p. 925], the semigroup $\langle \gamma, \delta \rangle$ has

$$\frac{1}{2}(\delta - 1)(\gamma - 1) \tag{2.14}$$

gaps. Let $\overline{\mathcal{C}}$ denote the curve defined by $F(x_1, x_2) = 0$ over $\overline{\mathbb{F}}_q$, then it follows from the Weierstrass Gap Theorem [63, I.6.7] that the genus of $\overline{\mathcal{C}}$ equals the number of gaps of the Weierstrass semigroup of $\bar{P}_\infty$, i.e. the quantity in (2.14). Finally, by [63, III.6.3(b)] we have that the genus of $\overline{\mathcal{C}}$ equals that of $\mathcal{C}$, and hence the lemma follows. □

Armed with the above two lemmas, we are now ready to prove the promised description of the ring $\mathcal{R}$.

Proposition 2.19. *It holds that*

1. *The Weierstrass semigroup of P_∞ is $H(P_\infty) = \langle \gamma, \delta \rangle$.*
2. *$\mathcal{R} = \mathbb{F}_q[x_1, x_2]$.*

Proof. We prove each item of the proposition separately.

1. Arguing as in the proof of Lemma 2.17 we have that $\mathbb{F}_q[x_1, x_2] \subseteq \mathcal{R}$, and by Proposition 2.16 this implies that

$$\langle \gamma, \delta \rangle \subseteq H(P_\infty). \tag{2.15}$$

By the Weierstrass Gap Theorem we know that the number of gaps in the semigroup on the right hand side of Equation (2.15) is equal to the genus of $\mathcal{C}$, and by Lemma 2.18 we know that this quantity is $\frac{1}{2}(\gamma - 1)(\delta - 1)$. On the other hand we known from [52, p. 925] that the semigroup on the left hand side of Equation (2.15) has the same number of gaps, and hence we conclude that $\langle \gamma, \delta \rangle = H(P_\infty)$.

2. As argued above we have that $\mathbb{F}_q[x_1, x_2] \subseteq \mathcal{R}$ and hence it suffices to prove the reverse inclusion. To this end, let $f \in \mathcal{R}$ and let $a = -v_{P_\infty}(f)$. By item 1 of this proposition, we know that a must be of the form $a = \gamma i_1 + \delta j_1$. This means that there exist an element $\alpha_1 \in \mathbb{F}_q$ such that $f_1 = f - \alpha_1 x_1^{i_1} x_2^{j_1}$ satisfies $-v_{P_\infty}(f_1) < a$. Since $\mathbb{F}_q[x_1, x_2] \subseteq \mathcal{R}$, f_1 is an element of $\mathcal{R}$, and hence we may apply the above argument recursively, to get that there exist $\alpha_1, \ldots, \alpha_u$ and $(i_1, j_1), \ldots, (i_u, j_u)$ such that
$$f' = f - \left(\sum_{h=1}^{u} \alpha_h x_1^{i_h} x_2^{j_h} \right), \tag{2.16}$$
satisfies $-v_{P_\infty}(f') \leq 0$. Furthermore f' is an element of $\mathcal{R}$ and hence it can only have poles at P_∞. This means that f' has no poles at all, and hence that it is a constant in $\mathbb{F}_q$. By Equation (2.16) this implies that f lies in $\mathbb{F}_q[x_1, x_2]$ as desired.

□

We conclude the section by giving an example of a well-known member of the class of simple C_{ab} curves.

Example 2.20 (Hermitian curve)**.** Let $q = r^2$ for some prime power r, then the Hermitian curve $\mathcal{H}$ over $\mathbb{F}_q$ can be defined by the polynomial $F(X_1, X_2) = X_2^r + X_2 - X_1^{r+1}$ or equivalently by the equation
$$x_2^r + x_2 = x_1^{r+1}.$$
We see that this relation satisfies item 1 in Definition 2.14 and furthermore, since $\gcd(\gamma, \delta) = \gcd(r, r+1) = 1$, item 2 is satisfied too. Finally, since $\frac{\partial F}{\partial X_2} = 1 \neq 0$ the curve is non-singular, and thus $\mathcal{H}$ satisfies all the requirements to be a simple C_{ab} curve. The Hermitian curve is especially interesting for us since it has many points satisfying Assumption 2.1. More specifically, all the rational places $Q_1, \ldots, Q_{r^2}$ of $\mathbb{F}_q(x_1)$ different from the pole of x_1, split completely in the extension $\mathbb{F}_q(x_1, x_2)/\mathbb{F}_q(x_1)$ and therefore $\mathcal{H}$ has r^3 places satisfying the assumption. We will return to the Hermitian curve in

Example 4.9 where we will use it to construct good algebraic-geometry codes, and show that the interpolation step of the G–S algorithm for these curves, can be done efficiently.

2.4 Key-equations for simple C_{ab} curves

In this section we will use Theorem 2.12 to derive a set of key-equations characterizing all polynomials satisfying the interpolation constraints in Problem 2.8.

2.4.1 Changing base ring

Let notation be as in Theorem 2.12. The theorem shows that $Q(\mathbf{T}) = \sum_{\mathbf{j}\in\Delta_\ell} q_\mathbf{j}\mathbf{T}^\mathbf{j}$ is an interpolation polynomial if and only if the coefficient vector $\mathbf{q} = (q_\mathbf{j})_{\mathbf{j}\in\Delta_\ell}$ is an element in the $\mathcal{R}$–column span of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. We would like to use this fact to develop an algorithm for computing valid interpolation polynomials. A natural first attempt at this, is to make an algorithm that proceeds by repeatedly making column operations that cancel leading terms (with respect to $\deg_\mathbf{w}$) in the columns of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. However, it does *not* hold that for $A, B \in \mathcal{R}$, $\deg_\mathbf{w}(A) \leq \deg_\mathbf{w}(B)$ implies that A divides B in $\mathcal{R}$. Hence it may not always be possible to cancel leading terms of the columns of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. Thus, without modifications, such an attempt will not yield the desired algorithm.

To get an algorithmically more convenient formulation of the interpolation problem, we aim for an extension of Theorem 2.12 which characterizes the coefficient vectors of interpolation polynomials as elements in the $\mathbb{F}_q[x_1]$–column span of an explicit matrix. Below we will obtain such a result by changing the base ring of the $\mathcal{R}$–module spanned by the columns of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. In Proposition 2.19 we saw that $\mathcal{R} = \mathbb{F}_q[x_1, x_2]$ and hence

$$\mathcal{R} \simeq \mathbb{F}_q[x_1]^\gamma. \tag{2.17}$$

This means that any $A \in \mathcal{R}$ can be written uniquely on the form $A = \sum_{i=0}^{\gamma-1} a_i(x_1)x_2^i$. We can make the isomorphism in Equation (2.17) explicit by defining

$$\rho(A) = (a_0, \ldots, a_{\gamma-1}) \in \mathbb{F}_q[x_1]^\gamma. \tag{2.18}$$

Furthermore, we can extend this to an identification of $\mathcal{R}^m$ and $\mathbb{F}_q[x_1]^{\gamma m}$, by for a vector $\mathbf{A} = (A_1, \ldots, A_m) \in \mathcal{R}^m$ letting

$$\rho(\mathbf{A}) = (\rho(A_1), \ldots, \rho(A_m)) \in \mathbb{F}_q[x_1]^{\gamma m}. \tag{2.19}$$

We now use this identification to "expand" $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$ to a matrix with entries in $\mathbb{F}_q[x_1]$.

Definition 2.21. For given $E \in \mathcal{R}$ and $\mathbf{R} \in \mathcal{R}^v$, the matrix $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ is defined as follows:

- $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ is an $\gamma m_\ell \times \gamma m_\ell$ matrix with entries in $\mathbb{F}_q[x_1]$,
- The rows and columns of $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ are indexed by $(i', \mathbf{j}')$ and $(i, \mathbf{j})$ respectively, both running in $\{0, \ldots, \gamma - 1\} \times \Delta_\ell$.
- The $(i, \mathbf{j})$-th column of $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ is given by

$$(\mathbf{B}_{s,\ell}(\mathbf{R}, E))_{(i,\mathbf{j})} = \rho\left(x_2^i \cdot (\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}\right). \tag{2.20}$$

Note that in the special case $\gamma = 1$, no expansion occurs and $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ is equal to $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. Using this matrix, we now get the following result.

Proposition 2.22. *A polynomial*

$$Q(\mathbf{T}) = \sum_{i=0}^{\gamma-1} \sum_{\mathbf{j} \in \Delta_\ell} q_{i,\mathbf{j}}(x_1) x_2^i \mathbf{T}^{\mathbf{j}} \in \mathcal{R}[\mathbf{T}]$$

is an interpolation polynomial of total degree in $\mathbf{T}$ *at most* ℓ*, if and only if the coefficient vector* $\mathbf{q} = (q_{i,\mathbf{j}})$ *is contained in the* $\mathbb{F}_q[x_1]$*-module spanned by the columns of* $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$*.*

Proof. Let M be the $\mathcal{R}$–module spanned by the columns of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. By Proposition 2.19 the ring $\mathcal{R}$ is a finitely generated free module over $\mathbb{F}_q[x_1]$ with basis $\{1, x_2, \ldots, x_2^{\gamma-1}\}$. Therefore M is also a free module over $\mathbb{F}_q[x_1]$ with basis

$$x_2^i \cdot (\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}, \quad 0 \le i \le \gamma - 1 \text{ and } \mathbf{j} \in \Delta_\ell. \tag{2.21}$$

Here $(\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}$ denotes the $\mathbf{j}$-th column of $\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)$. Note that the elements in Equation (2.21) correspond to the columns of $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$. Therefore the proposition follows from Theorem 2.12. □

2.4.2 Identifying weights

Proposition 2.22 above characterizes all interpolation polynomials. To get the desired key-equations for the interpolation problem in Problem 2.8, we also need to characterize *valid* interpolation polynomials. To this end, we need the following definition.

Definition 2.23 (Weighted degree)**.** Let $\mathbf{w} = (w_0, w_1, \dots, w_m) \in \mathbb{N} \times \mathbb{Z}^m$ be a vector of *weights*. Then we define the following weighted degree of a vector $\mathbf{a} = (a_1, \dots, a_m) \in \mathbb{F}_q[x_1]^m$,

$$\deg_{\mathbf{w}}(\mathbf{a}) = \begin{cases} \max_{\substack{i=1,\dots,m \\ a_i \neq 0}} \{w_0 \cdot \deg(a_i) + w_i\} & \text{if } \mathbf{a} \neq \mathbf{0}, \\ -\infty & \text{if } \mathbf{a} = \mathbf{0}. \end{cases}$$

Furthermore, for an $m \times m$ matrix $\mathbf{A}$ with columns $\mathbf{A}_1, \dots, \mathbf{A}_m$ and entries in $\mathbb{F}_q[x_1]$, we define its weighted degree to be

$$\deg_{\mathbf{w}}(\mathbf{A}) = \sum_{\substack{j=1,\dots,m \\ \mathbf{A}_j \neq 0}} \deg_{\mathbf{w}}(\mathbf{A}_j).$$

In Definition 2.6 we defined a weighted degree of multivariate polynomials in $\mathcal{R}[\mathbf{T}]$, also denoted $\deg_{\mathbf{w}}$. In the following it will be clear from the context which of the two versions of the weighted degree is meant. Furthermore, in special cases the weighted degree of a polynomial *agrees* with the weighted degree of its coefficient vector as the following result shows.

Proposition 2.24. *Let* $\mathbf{q}$ *be a vector in* $\mathcal{R}^{m_\ell}$ *and let* $\mathbf{w} \in \mathbb{Z}^v$ *be a weight vector. Furthermore, let*

$$\rho(\mathbf{w}) = (\gamma, \mathbf{w}') \in \mathbb{N} \times \mathbb{Z}^{\gamma m_\ell}, \tag{2.22}$$

where $\mathbf{w}'$ *is indexed by* $(i, \mathbf{j})$ *in* $\{0, \dots, \gamma - 1\} \times \Delta_\ell$*, and* $[\mathbf{w}']_{i,\mathbf{j}} = \delta i + \mathbf{w} \cdot \mathbf{j}$*. For a polynomial*

$$Q(\mathbf{T}) = \sum_{\mathbf{j} \in \Delta_\ell} q_{\mathbf{j}} \mathbf{T}^{\mathbf{j}} = \sum_{\mathbf{j} \in \Delta_\ell} \sum_{i=0}^{\gamma-1} q_{i,\mathbf{j}}(x_1) x_2^i \mathbf{T}^{\mathbf{j}}$$

in $\mathcal{R}[\mathbf{T}]$ *it holds that*

$$\deg_{\mathbf{w}}(Q(\mathbf{T})) = \deg_{\rho(\mathbf{w})}(\rho(\mathbf{q})) = \max_{(i,\mathbf{j}), q_{i,\mathbf{j}} \neq 0} \{\gamma \deg(q_{i,\mathbf{j}}) + i\delta + \mathbf{w} \cdot \mathbf{j}\}.$$

Proof. By definition of the weighted degree of $Q(\mathbf{T})$, the proposition will follow if we can show that

$$-v_{P_\infty}(q_{\mathbf{j}}) = \max_{i, q_{i,\mathbf{j}} \neq 0} \{\gamma \deg(q_{i,\mathbf{j}}) + i\delta\}.$$

We have that

$$-v_{P_\infty}(q_{\mathbf{j}}) = -v_{P_\infty}\left(\sum_{i=0}^{\gamma-1} q_{i,\mathbf{j}} x_2^i\right). \tag{2.23}$$

By an argument similar to the one in the proof Proposition 2.18, we have that all terms in the sum on the right hand side of this expression, have distinct valuation at P_∞. Hence, by the Strict Triangle Inequality [63, I.1.10], it follows that

$$\begin{aligned} -v_{P_\infty}(q_{\mathbf{j}}) &= -v_{P_\infty}\left(\sum_{i=0}^{\gamma-1} q_{i,\mathbf{j}} x_2^i\right) \\ &= \max_{i, q_{i,\mathbf{j}} \neq 0} \left\{-v_{P_\infty}(q_{i,\mathbf{j}}) - v_{P_\infty}\left(x_2^i\right)\right\} \\ &= \max_{i, q_{i,\mathbf{j}} \neq 0} \{\gamma \deg(q_{i,\mathbf{j}}) + i\delta\}. \end{aligned}$$

As noted above, this proves the proposition. □

In words the above proposition says that if we identify a multivariate polynomial in $\mathcal{R}[\mathbf{T}]$ of total degree at most ℓ, with its coefficient vector in $\mathcal{R}^{m_\ell}$, and then identify this with a vector $\mathbb{F}_q[x_1]^{\gamma m_\ell}$ using the isomorphism ρ in (2.19), then the weighted degree of the multivariate polynomial (with respect to $\mathbf{w}$) is equal to the weighted degree of the vector (with respect to $\rho(\mathbf{w})$). Using this fact in combination with Proposition 2.22 we can now prove the following final and algorithmically convenient set of key-equations characterizing valid interpolation polynomials.

Proposition 2.25 (Key-equations)**.** *Let* $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ *be the matrix from Definition 2.21 and let* $\mathbf{q} = \mathbf{B}_{s,\ell}(\mathbf{R}, E)\mathbf{c}$ *for some vector* $\mathbf{c} = (c_{i,\mathbf{j}})$ *in* $\mathbb{F}_q[x_1]^{\gamma m_\ell}$. *Then*

$$Q(\mathbf{T}) = \sum_{i=0}^{\gamma-1} \sum_{\mathbf{j} \in \Delta_\ell} q_{i,\mathbf{j}} x_2^i \mathbf{T}^{\mathbf{j}}, \tag{2.24}$$

is a valid interpolation polynomial if and only if $\deg_{\rho(\mathbf{w})}(\mathbf{q}) < \Delta$.

Proof. From Proposition 2.24 we have that $\deg_{\mathbf{w}}(Q) = \deg_{\rho(\mathbf{w})}(\mathbf{q})$. Therefore, since Proposition 2.22 gives that any $Q(\mathbf{T})$ of the form (2.24) is an interpolation polynomial, the proposition follows. □

2.5 Key-equations for Reed–Solomon codes

In this section we show that Reed–Solomon codes are a special case of algebraic-geometry codes defined over simple C_{ab} curves. Furthermore, we specialize the key-equations in Proposition 2.25 to this particular case, and we show that the resulting key-equations agree with those known in the literature. In particular we will show that in the light of Proposition 2.25 the (seemingly unrelated) key-equations for Reed–Solomon codes in [4] and [43] in fact only differ by a simply rewriting. Furthermore, we show that the "re-encoding" technique from [39] fits naturally into our key-equations.

2.5.1 Key-equations

The function field corresponding to the projective line $\mathbb{P}^1_{\mathbb{F}_q}$ is the rational function field $\mathbb{F}_q(\mathbb{P}^1_{\mathbb{F}_q}) = \mathbb{F}_q(x_1)$. Somewhat artificially, we may view $\mathbb{F}_q(\mathbb{P}^1_{\mathbb{F}_q})$ as the trivial extension of $\mathbb{F}_q(x_1)$ defined by the equation $x_2 = 1$, and when looked at in this way, we see that $\mathbb{P}^1_{\mathbb{F}_q}$ satisfies all the requirements in Definition 2.14 to be a simple C_{ab} curve. The rational places of $\mathbb{F}_q(x_1)$ different from P_∞ are in bijective correspondence with the elements of $\mathbb{F}_q$ [63, I.2.3], and thus for $n \leq q$ it is possible to choose places $P_1, \ldots, P_n$ satisfying Assumption 2.1. Since $x_2 = 1$, we get that for the projective line the ring $\mathcal{R}$ is especially simple, namely $\mathcal{R} = \mathbb{F}_q[x_1, x_2] = \mathbb{F}_q[x_1]$.

Let $P_1, \ldots, P_n$ be places of the projective line, different from P_∞ and let C be the Reed–Solomon code over $\mathbb{F}_q$ of length $n \leq q$ and dimension k,

$$C = \{(f(P_1), \ldots, f(P_n)) \mid f \in \mathbb{F}_q[x_1], \deg(f) < k\}.$$

Let $\mathbf{y} = (y_1, \ldots, y_n)$ be a received word and say that we wish to correct τ errors with the G–S algorithm. Furthermore, let $s \geq 1$ be a multiplicity parameter, and let $\ell \geq s$ be a degree bound. In Section 1.2.2 we saw that in the interpolation step of the G–S algorithm we need to find a non-zero polynomial $Q \in \mathbb{F}_q[x_1, T]$ such that:

1. Q has a root of multiplicity at least s in the points (P_i, y_i) for $1 \leq i \leq n$.
2. Q must be such that $\deg_T(Q) \leq \ell$ and $\deg_{k-1}(Q) < s(n - \tau)$.

For details on the origins of these requirements, see Section 4.2.1. The interpolation requirements in the above two items can be formulated in the language of Problem 2.8, by letting $v = 1$ (so that $\mathbf{T} = T$), keeping the multiplicity parameter and the degree bound, and by letting:

1. The interpolation places be $\mathbf{P} = (P_1, \ldots, P_n)$, and the interpolation values be $\mathbf{Y} = \mathbf{y}$.
2. The weighted degree bound be $\Delta = s(n - \tau)$.

Therefore we can specialize Proposition 2.25 to the following set of key-equations for list decoding of Reed–Solomon codes:

Corollary 2.26 (Key-equations for Reed–Solomon codes)**.** *Let notation be as above. Furthermore, let $R \in \mathbb{F}_q[x_1]$ be the Lagrangian interpolation polynomial satisfying $R(P_i) = y_i$ for $1 \leq i \leq n$, and let $E = \prod_{i=1}^{n}(x_1 - x_1(P_i))$. Then*

$$Q(T) = \sum_{j=0}^{\ell} q_j(x_1)T^j \in \mathbb{F}_q[x_1, T], \tag{2.25}$$

is an interpolation polynomial, if and only if there exist a vector $\mathbf{c} \in \mathbb{F}_q[x_1]^{\ell+1}$ such that $\mathbf{q} = \mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)\mathbf{c}$. If furthermore

$$\deg_{\rho(k-1)}(\mathbf{q}) < s(n - \tau), \tag{2.26}$$

then $Q(T)$ is also a valid interpolation polynomial.

Note that

$$\rho(k-1) = (1, 0, k-1, 2(k-1), \ldots, \ell(k-1)),$$

and therefore if we use the weighted degree from Definition 2.6, then the degree requirement in Equation (2.26) can be formulated as

$$\deg_{k-1}(\mathbf{q}) < s(n - \tau).$$

The rows of the equation $\mathbf{q} = \mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)\mathbf{c}$ in the above corollary give rise to $\ell + 1$ linear equations with coefficients in $\mathbb{F}_q[x_1]$. It is these equations that we regard as the *key-equations* for Reed–Solomon codes. These key-equations were derived in this form in [5], building on the work in [4, 43]. To illustrate the corollary we now give an example of the interpolation step in the G–S algorithm for Reed–Solomon codes.

Example 2.27. This example is taken from [38, p. 133]. Consider a $(15, 7, 9)$ Reed–Solomon code over $\mathbb{F}_{16}$ and let α be a primitive element of this field, satisfying $\alpha^4 + \alpha + 1$. Since the minimum distance of the code is 9 it can correct four errors. However, by selecting $s = 4$ and $\ell = 6$, five errors can be corrected using list decoding. Let the information polynomial be

$f(x_1) = x_1^6 + x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1 + 1$, and say we receive the following corrupted word

$$\mathbf{y} = (0, \alpha^5, \alpha^{10}, \alpha^3, \alpha^5, 1, \alpha^6, \alpha^7, \alpha^{10}, \alpha^9, 1, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}).$$

We have that $E = \prod_{i=0}^{14}(x_1 - \alpha^i) = x_1^{15} - 1$ and by Lagrangian interpolation, we find $R = x_1^{10} + x_1^6 + x_1^4 + x_1^3 + x_1^2 + x_1$. Furthermore, we have

$$\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E) = \begin{bmatrix} E^4 & -E^3R & E^2R^2 & -ER^3 & R^4 & -R^5 & R^6 \\ 0 & E^3 & -2E^2R & 3ER^2 & -4R^3 & 5R^4 & -6R^5 \\ 0 & 0 & E^2 & -3ER & 6R^2 & -10R^3 & 15R^4 \\ 0 & 0 & 0 & E & -4R & 10R^2 & -20R^3 \\ 0 & 0 & 0 & 0 & 1 & -5R & 15R^2 \\ 0 & 0 & 0 & 0 & 0 & 1 & -6R \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In order for the G–S algorithm to be successful, we need to find an interpolation polynomial of weighted degree strictly less than $s(n - \tau) = 40$. Corollary 2.26 shows that this is equivalent to finding a vector in the $\mathbb{F}_q[x_1]$–column span of the above matrix of weighted degree strictly less than 40. Later we will get an efficient method for finding such a vector, but for now we will be content with noting that if we let $\mathbf{c}$ be the vector

$$\mathbf{c} = \left(1, 0, x_1^{10}, 0, x_1^{20}, 0, 1\right)^T,$$

and take $\mathbf{q} = \mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)\mathbf{c}$, then we get the following interpolation polynomial

$$\begin{aligned} Q(T) &= \sum_{j=0}^{6} q_i(x_1)T^j \\ &= T^6 + \left(x_1^{12} + x_1^8 + x_1^6 + x_1^4 + x_1^2\right)T^4 + \\ &\quad \left(x_1^{24} + x_1^{16} + x_1^{12} + x_1^{10} + x_1^8 + x_1^4\right)T^2 + \\ &\quad \left(x_1^{36} + x_1^{32} + x_1^{26} + x_1^{20} + x_1^{18} + x_1^{14} + x_1^{12} + x_1^8 + x_1^6 + 1\right). \end{aligned}$$

The weighted degree of this polynomial is $\max_j\{\deg(q_j(x_1)) + 6j\} = 36$ and since this is strictly smaller than $s(n - \tau)$ we are guaranteed that $T - f(x_1)$ is a factor of $Q(T)$. It can be checked that

$$\begin{aligned} Q(T) &= (T - (x_1^6 + x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1 + 1))^2 \cdot \\ &\quad (T - (x_1^6 + \alpha^5 x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1 + \alpha^{10}))^2 \cdot \\ &\quad (T - (x_1^6 + \alpha^{10} x_1^5 + x_1^4 + x_1^3 + x_1^2 + x_1 + \alpha^5))^2, \end{aligned}$$

and thus we indeed get the expected factor.

2.5.2 Exploiting structure in an interpolation problem

The interpolation problem in the G–S algorithm for a Reed–Solomon code of length n and multiplicity parameter s, may be directly formulated and solved as a linear algebra problem, see Proposition 5.5 or [30] for details. The matrix of the resulting system has about $\left(\binom{s+1}{2}n\right)^2 = \mathcal{O}\left(s^4n^2\right)$ entries, and using Gaussian elimination it may be solved in complexity $\mathcal{O}\left(s^6n^3\right)$. However, since the linear system results from an interpolation problem, it contains *structure* which is not exploited by the Gaussian elimination approach. In [4] it was shown that the interpolation problem in the G–S algorithm can be equivalently described by a set of key-equations. Furthermore, it was shown that those key-equations can be formulated as a linear system on block Hankel form, with $\mathcal{O}\left(s^3n\right)$ distinct entries (if the common element in the anti-diagonals are counted as a single entry). See also [8] for a direct derivation of the block Hankel system, without the use of key-equations. One way to interpret this result is that the key-equations exploit the redundancy of the original interpolation problem to "compress" the problem by a factor of $\mathcal{O}\left(sn\right)$. Fortunately this compressed problem is also easier to solve, and for instance in [58] an extension of the Berlekamp–Massey algorithm from [56] is used to solve the block Hankel system in complexity $\mathcal{O}\left(s^5n^2\right)$, again saving a factor of $\mathcal{O}\left(sn\right)$ when compared to the direct linear algebra approach.

In [49] Olshevsky and Shokrollahi used a different approach for exploiting the structure of the interpolation problem in the G–S algorithm. Using the concept of *displacement rank* they showed how the linear equation system resulting from the interpolation problem, can be considered "sparse", and furthermore they showed how this sparsity allows one to solve the system in complexity $\mathcal{O}\left(s^5n^2\right)$. Note that this matches the performance of the algorithm mentioned above.

2.5.3 Comparison to other key-equations

Let R and E be as in Corollary 2.26. In this notation the key-equations derived in [4, Prop. 1] state that $Q(T) = \sum_{j=0}^{\ell} q_j(x_1)T^j$ is an interpolation polynomial if and only if

$$E(x_1)^{s-b} \mid \mathcal{H}_T^{(b)}\left(Q\right)(x_1, R(x_1)),$$

for $0 \leq b < s$. In our matrix notation, this is equivalent to the existence of a vector $\mathbf{c} = (c_0, \ldots, c_\ell) \in \mathbb{F}_q[x_1]^{\ell+1}$ such that $\mathbf{A}_\ell(R)\mathbf{q} = \mathbf{D}_{s,\ell}(E)\mathbf{c}$. By Proposition 2.11 this is furthermore equivalent to $\mathbf{q} = \mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)\mathbf{c}$. The latter is exactly the key-equations in Corollary 2.26 and we thus conclude that our key-equations only differ from those in [4, Prop. 1] by a single matrix inversion.

In [43] a different set of key-equations for the interpolation step in the Guruswami–Sudan list decoder, was derived. If we let

$$A_j = \begin{cases} E(x_1)^{s-j}(T - R(x_1))^j & 0 \leq j \leq s \\ T^{j-s}(T - R(x_1))^s & s < j \leq \ell \end{cases},$$

then [43, Cor. 4] states that $Q(T)$ in an interpolation polynomial if and only if it is of the form $Q(T) = \sum_{j=0}^{\ell} c_j(x_1)A_j$. Note that each of the polynomials A_j can be expressed as an $\mathbb{F}_q[x_1]$–linear combination of the polynomials

$$B_j = E(x_1)^{\max(0,s-j)}(T - R(x_1))^j, \quad 0 \leq j \leq \ell$$

and hence we also have that $Q(T)$ is an interpolation polynomial if and only if it is of the form $Q(T) = \sum_{j=0}^{\ell} c_j(x_1)B_j$. We have that

$$\mathcal{H}_T^{(i)}(B_j)\Big|_{T=0} = E(x_1)^{\max(0,s-j)}\binom{j}{i}(-R(x_1))^{j-i},$$

and thus the coefficient to T^i in B_j is exactly the (i,j)-th entry of the matrix $\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)$. Therefore we get that [43, Cor. 4] is equivalent to the statement that $Q(T)$ is an interpolation polynomial if and only if $\mathbf{q}$ is of the form $\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)\mathbf{c}$. This last statement is exactly Corollary 2.26, and we conclude that our key-equations only differ by a simple rewritings from those in [43]. This in turn means that the apparently different key-equations in [4] and [43], are in fact only a matrix inversion and a simple rewriting apart.

2.5.4 Re-encoding

In [39] Ahmed, Kötter, Ma and Vardy suggest a simple but powerful technique, known as *re-encoding*, for rewriting the key-equations for Reed–Solomon codes, into an equivalent set of equations involving polynomials of lower degree. For some algorithms this rewritten problem is much easier to solve than the original one. In this section we show how the re-encoding technique can be naturally understood in terms of the key-equations in Corollary 2.26.

Let $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}_q^n$ be a received word, and let $g(x_1)$ be the Lagrange interpolation polynomial of degree at most $k-1$ such that $g(P_i) = y_i$ for $1 \leq i \leq k$. Consider the vector

$$\mathbf{y}' = (y_1 - g(P_1), \ldots, y_n - g(P_n)) = \left(0, \ldots, 0, y'_{k+1}, \ldots, y'_n\right). \tag{2.27}$$

If we can find a polynomial $f(x_1)$ of degree at most $k-1$ such that the codeword $(f(P_1),\dots,f(P_n))$ agrees with $\mathbf{y}'$ in at least $n-\tau$ positions, then $(f(P_1)+g(P_1),\dots,f(P_n)+g(P_n))$ agrees with $\mathbf{y}$ in least $n-\tau$ positions. Therefore, when list decoding a Reed–Solomon code we can translate (or re-encode) the interpolation problem, so that the first k positions of the received word are all zero. This in turn means that the polynomial R in Corollary 2.26 is divisible by

$$E_0(x_1) = \prod_{i=1}^{k}(x_1 - x_1(P_i)).$$

By definition of $\mathbf{A}_\ell$ and $\mathbf{D}_{s,\ell}$ this implies that each entry in the i-th row of $\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)$ is divisible by $E_0^{\max(s-i,0)}$. We can write this fact as the following matrix-factorization

$$\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E) = \mathbf{D}_{s,\ell}(E_0)\mathbf{K}\mathbf{D}_{s,\ell}(E/E_0), \tag{2.28}$$

where $\mathbf{K}$ is the $\ell+1\times\ell+1$ matrix defined by

$$[\mathbf{K}]_{i,j} = \begin{cases} \binom{j}{i}(-R/E_0)^{j-i} & 0\le i\le \ell \text{ and } 0\le j\le s \\ \binom{j}{i}(-R/E_0)^{s-i}(-R)^{j-s} & 0\le i\le s \text{ and } s<j\le \ell \\ \binom{j}{i}(-R)^{j-i} & s<i\le\ell \text{ and } s<j\le\ell \end{cases}$$

Note that the the left most $s+1$ columns of $\mathbf{K}$ equal those of $\mathbf{A}_\ell(-R/E_0)$, and that the bottom most $\ell-s$ rows equal those of $\mathbf{A}_\ell(-R)$. Equation (2.28) contains the essence of the re-encoding technique. Let

$$(1,\mathbf{w}) = (1,w_0,\dots,w_\ell) \in \mathbb{N}_0\times\mathbb{Z}^{\ell+1},$$

be a weight vector as in Definition 2.23, and let $\mathbf{c}$ be a vector in $\mathbb{F}_q[x_1]^{\ell+1}$. Then Equation (2.28) shows that $\mathbf{q} = \mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)\mathbf{c}$ satisfies $\deg_{1,\mathbf{w}}(\mathbf{q}) < \Delta$ if and only if

$$\mathbf{q}' = \mathbf{K}\mathbf{D}_{s,\ell}(E/E_0)\mathbf{c}, \tag{2.29}$$

satisfies $\deg_{1,\widetilde{\mathbf{w}}}(\mathbf{q}') < \Delta$, where $\widetilde{\mathbf{w}}$ is the weight vector defined by

$$\widetilde{w}_i = w_i + \deg\left(E_0^{\max(s-i,0)}\right) = w_i + k\max(s-i,0),$$

for $0\le i\le\ell$. In the interpolation problem for Reed–Solomon codes we have $\Delta = s(n-\tau)$, and the weighting of the rows of $\mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)$ corresponds to taking $w_i = i(k-1)$. Therefore

$$\widetilde{w}_i = i(k-1) + k\max(s-i,0) = \begin{cases} sk - i & \text{if } 0\le i\le s, \\ i(k-1) & \text{if } s<i\le\ell. \end{cases}$$

Note that $\widetilde{w}_i \geq s(k-1)$ for $0 \leq i \leq \ell$. Therefore the vector $\mathbf{q}'$ from Equation (2.29) satisfy $\deg_{1,\widetilde{\mathbf{w}}}(\mathbf{q}') < \Delta = s(n-\tau)$ if and only if

$$\deg_{1,\mathbf{w}'}(\mathbf{q}') < s(n-\tau) - s(k-1) = s(d-\tau),$$

where $d = n-k+1$ is the minimum distance of C, and $\mathbf{w}'$ is the weight vector defined by

$$w'_i = \begin{cases} s-i & \text{if } 0 \leq i \leq s, \\ (i-s)(k-1) & \text{if } s < i \leq \ell. \end{cases} \tag{2.30}$$

Summarizing the above translations of the interpolation problem, we have proved the following alternative set of key-equations for Reed–Solomon codes.

Proposition 2.28. *Let $\mathbf{y} \in \mathbb{F}_q^n$ be a received word and let*

$$\mathbf{y}' = (y'_1, \ldots, y'_n) = (0, \ldots, 0, y'_{k+1}, \ldots, y'_n) \in \mathbb{F}_q^n,$$

be the translated word from Equation (2.27). Let $E = \prod_{i=1}^n (x_1 - x_1(P_i))$ and let R be the Lagrangian interpolation polynomial such that $R(P_i) = y'_i$ for $1 \leq i \leq n$. Let $\mathbf{P} = (P_1, \ldots, P_n)$. Then $Q(T) = \sum_{j=0}^{\ell} q_j(x_1)T^j$ is an interpolation polynomial for $(\mathbf{P}, \mathbf{y}', s, \ell)$ if and only if it is of the form

$$Q(T) = \sum_{j=0}^{\ell} q'_j(x_1) E_0(x_1)^{\max(0,s-j)} T^j, \tag{2.31}$$

and there exists a vector $\mathbf{c} \in \mathbb{F}_q[x_1]^{\ell+1}$ such that $\mathbf{q}' = \mathbf{K}\mathbf{D}_{s,\ell}(E/E_0)\mathbf{c}$. Furthermore, let $Q(T)$ be as in Equation (2.31) and let $\mathbf{w}'$ be the weight vector defined in Equation (2.30). Then $Q(T)$ is a valid interpolation polynomial, i.e. $\deg_{k-1}(Q(T)) < s(n-\tau)$, if and only if $\deg_{1,\mathbf{w}'}(\mathbf{q}') < s(d-\tau)$.

The advantage of the rewritten (or re-encoded) key-equations in Proposition 2.28 over those in Corollary 2.26, is that they involve polynomials of lower degree. For some algorithms, the degrees of the polynomials R and E from the key-equations in Corollary 2.26 determine the algorithm's complexity. In the re-encoded key-equations above, the roles that $\mathbf{A}_\ell(-R)$ and $\mathbf{D}_{s,\ell}(E)$ play in Corollary 2.26, are played by $\mathbf{K}$ and $\mathbf{D}_{s,\ell}(E/E_0)$ respectively. Looking at the definition of $\mathbf{K}$ we see that one may view this matrix as a modification of $\mathbf{A}_\ell(-R)$, in which "most" occurrences of R are replaced by R/E_0. While R and E both have degree at most n, the polynomials R/E_0 and E/E_0 both have degree at most $n-k$. Thus for algorithms whose complexity depends on the degrees of R and E, the re-encoding technique allows one to, rougly speaking, replace n by $n-k$ in their complexity estimates. The linear algebra approach mentioned in Section 2.5.2 and the Lee–O'Sullivan algorithm [43] are examples of algorithms with this property, see [39, 45] for details. For a fixed rate R, re-encoding does not change the asymptotic complexity of these two algorithms (since the ratio of $n - k$ and n is the constant $1 - R$). However, re-encoding significantly improves the practical performance of these algorithms, especially for high rate codes.

Chapter 3

The short-basis algorithm

Given a module, what is its “shortest” possible basis, and can we compute this basis efficiently? In this chapter we will answer this question, when the shortness of a vector is measured with respect to the weighted degree $\deg_{\mathbf{w}}$ from Definition 2.23, and the base ring of the module is a univariate polynomial ring. We do so by presenting a *short-basis algorithm* which given a basis $\mathbf{A}$ of an $\mathbb{F}_q[x_1]$–module M, a weight vector $\mathbf{w}$ and a target t, computes another basis $\mathbf{A}'$ of M such that either $\deg_{\mathbf{w}}(\mathbf{A}') \leq \deg_{\mathbf{w}}(\mathbf{A}) - t$, or $\mathbf{A}'$ is *reduced*. The presented algorithm uses a divide–and–conquer approach, similar to the one used in the fast euclidean algorithm for polynomials (see e.g. [20, p. 309]). It is an adaptation of an algorithm by Alekhnovich [2], generalized to compute short vectors with respect to weighted degrees.

In Proposition 2.25 it was shown that the interpolation problem posed in Problem 2.8 can be equivalently formulated as the problem of finding short vectors in an certain module. In Chapter 4 we use this to show how the short-basis algorithm can be applied to solve the interpolation problem. The material in this chapter is based on the paper [6].

3.1 The short-basis algorithm

In this section we present the short-basis algorithm. Before we can formulate the objectives of the algorithm, we need to establish some terminology.

3.1.1 Definitions and terminology

In the following $\mathbf{A}$ will denote an $m \times m$ matrix with entries in $\mathbb{F}_q[x_1]$ and we will denote its j-th column by $\mathbf{A}_j$. For the measure of "shortness" of a vector with entries in $\mathbb{F}_q[x_1]$ we will use the weighted degree from Definition 2.23. We now define the notions of *leading coordinate*, *critical position* and *reduced* with respect to this weighted degree.

Definition 3.1 (Leading coordinate, critical position, reduced)**.** Let

$$\mathbf{w} = (w_0, w_1, \ldots, w_m) \in \mathbb{N} \times \mathbb{Z}^m$$

be a weight vector and let $\mathbf{A}$ be an $m \times m$ matrix of polynomials in $\mathbb{F}_q[x_1]$.

- For a column $\mathbf{A}_j$ of $\mathbf{A}$, its *leading coordinate* with respect to $\deg_{\mathbf{w}}$, denoted $\mathrm{LC}_{\mathbf{w}}(\mathbf{A}_j)$, is the largest index i for which
$$w_0 \deg \mathbf{A}_{i,j} + w_i = \deg_{\mathbf{w}}(\mathbf{A}_j). \tag{3.1}$$
- A pair (i, j) satisfying Equation (3.1) will be called a *critical position*.
- A matrix $\mathbf{A}$ is said to be *reduced* with respect to $\deg_{\mathbf{w}}$ if for all pairs of distinct columns $\mathbf{A}_{j_1}$ and $\mathbf{A}_{j_2}$, it holds that $\mathrm{LC}_{\mathbf{w}}(\mathbf{A}_{j_1}) \neq \mathrm{LC}_{\mathbf{w}}(\mathbf{A}_{j_2})$.

The weighted degree $\deg_{\mathbf{w}}$ can be extended to a term order on $\mathbb{F}_q[x_1]^m$. Using the term–over–position extension (see [16, p. 211]), we get the following order:

Definition 3.2 (Term order $<_{\mathbf{w}}$, leading term)**.** Let $\mathbf{w} = (w_0, w_1, \ldots, w_m) \in \mathbb{N} \times \mathbb{Z}^m$ be a vector of weights. Let $\mathbf{e}_i$ denote the i-th unit vector in $\mathbb{F}_q[x_1]^m$, that is

$$\mathbf{e}_i = (\underbrace{0, \ldots, 0}_{i-1 \text{ times}}, 1, 0, \ldots, 0).$$

For two monomials $x_1^\alpha \mathbf{e}_i$ and $x_1^\beta \mathbf{e}_j$ in $\mathbb{F}_q[x_1]^m$, define $x_1^\alpha \mathbf{e}_i <_{\mathbf{w}} x_1^\beta \mathbf{e}_j$ if one of the following holds:

- $w_0\alpha + w_i < w_0\beta + w_j$,
- $w_0\alpha + w_i = w_0\beta + w_j$ and $i < j$.

A vector $\mathbf{A}_j$ in $\mathbb{F}_q[x_1]^m$ can be decomposed as a sum of monomials. We define the *leading term* of $\mathbf{A}_j$ with respect to $<_{\mathbf{w}}$, to be the largest monomial in this sum with respect to $<_{\mathbf{w}}$, and we denote it by $\mathrm{LT}_{\mathbf{w}}(\mathbf{A}_j)$.

To ease the language, we will in the following say that an $m \times m$ matrix $\mathbf{A}$ with entries in $\mathbb{F}_q[x_1]$, is a basis for an $\mathbb{F}_q[x_1]$–module M, if its *columns* form a basis for M.

3.1.2 Reduced bases and short bases

The following proposition gives a sufficient condition for when a basis of an $\mathbb{F}_q[x_1]$–module, is also a Gröbner basis.

Proposition 3.3. *Let* $\mathbf{A}$ *be a basis for the* $\mathbb{F}_q[x_1]$*–module* $M \subseteq \mathbb{F}_q[x_1]^m$*. If* $\mathbf{A}$ *is reduced with respect to* $\deg_{\mathbf{w}}$ *then* $\mathbf{A}$ *is a Gröbner basis for* M *with respect to* $<_{\mathbf{w}}$*.*

Proof. Assume that for any two distinct columns $\mathbf{A}_{j_1}$ and $\mathbf{A}_{j_2}$ we have $\mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_{j_1}\right) \neq \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_{j_2}\right)$. Then by definition of S–polynomials we have that for $j_1 \neq j_2$,

$$S(\mathbf{A}_{j_1}, \mathbf{A}_{j_2}) = 0.$$

By Buchberger's Criterion [16, p. 215], this means that $\mathbf{A}$ is a Gröbner basis for M with respect to $<_{\mathbf{w}}$. □

The result in the following proposition shows that among all bases of a module, the reduced bases are the shortest.

Proposition 3.4. *Let* $\mathbf{A}'$ *be a reduced basis for the module* $M \subseteq \mathbb{F}_q[x_1]^m$ *with respect to* $\deg_{\mathbf{w}}$*. For any basis* $\mathbf{A}$ *of* M *it holds that*

$$\deg_{\mathbf{w}}\left(\mathbf{A}'\right) \leq \deg_{\mathbf{w}}\left(\mathbf{A}\right).$$

Proof. Since $\mathbf{A}'$ is reduced we may reorder the columns of $\mathbf{A}'$ such that $\mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}'_j\right) = j$. Let $\mathbf{A}$ be some basis of M. We divide the proof into two cases, one where $\mathbf{A}$ is reduced, and one where it is not. First assume that $\mathbf{A}$ is reduced. Then, after reordering, we may assume that $\mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j\right) = j$. Since $\mathbf{A}'$ is a basis for M, there exist a vector $\mathbf{c} \in \mathbb{F}_q[x_1]^m$ such that

$$\mathbf{A}'\mathbf{c} = \mathbf{A}_1.$$

Furthermore, since $\mathbf{A}'$ is reduced with respect to $\deg_{\mathbf{w}}$, it is also a Gröbner basis with respect to $<_{\mathbf{w}}$ and hence $\mathrm{LT}_{\mathbf{w}}\left(c_i\mathbf{A}'_i\right) = \mathrm{LT}_{\mathbf{w}}\left(\mathbf{A}_1\right)$ for some i. For $c_j \neq 0$ it holds that $\mathrm{LC}_{\mathbf{w}}\left(c_j\mathbf{A}'_j\right) = \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}'_j\right)$ and hence we must have $i = 1$. This means that $\mathrm{LT}_{\mathbf{w}}\left(c_1\mathbf{A}'_1\right) = \mathrm{LT}_{\mathbf{w}}\left(\mathbf{A}_1\right)$, and in particular

$$\deg_{\mathbf{w}}\left(\mathbf{A}_1\right) = w_0\deg(\mathbf{A}_{1,1}) + w_1 \geq w_0\deg(\mathbf{A}'_{1,1}) + w_1 = \deg_{\mathbf{w}}\left(\mathbf{A}'_1\right).$$

Similarly one can prove that $\deg_{\mathbf{w}}\left(\mathbf{A}_j\right) \geq \deg_{\mathbf{w}}\left(\mathbf{A}'_j\right)$ for $1 \leq j \leq m$, and hence

$$\deg_{\mathbf{w}}\left(\mathbf{A}\right) \geq \deg_{\mathbf{w}}\left(\mathbf{A}'\right).$$

Next, assume that $\mathbf{A}$ is not reduced, and let $j_1 \neq j_2$ be such that $\mathrm{LC}_{\mathbf{w}}(\mathbf{A}_{j_1}) = \mathrm{LC}_{\mathbf{w}}(\mathbf{A}_{j_2})$. By symmetry we may assume that $\mathbf{A}_{j_1} \geq_{\mathbf{w}} \mathbf{A}_{j_2}$. This means that there exist $\alpha \in \mathbb{F}_q \setminus \{0\}$ and $\beta \geq 0$ such that

$$\alpha x_1^{\beta} \mathbf{A}_{j_2} + \mathbf{A}_{j_1} <_{\mathbf{w}} \mathbf{A}_{j_1}. \tag{3.2}$$

Let $\mathbf{U}$ denote the matrix such that $\mathbf{AU}$ is the matrix obtained by adding αx_1^{β} times the j_2-th column of $\mathbf{A}$ to j_1-th column. Such a matrix is called a transvection and has determinant 1. By definition of $\mathbf{U}$ we have that $\mathbf{A}^{(1)} = \mathbf{AU}$ is the matrix obtained by replacing the j_1-th column of $\mathbf{A}$ by $\alpha x_1^{\beta} \mathbf{A}_{j_2} + \mathbf{A}_{j_1}$, and since $\mathbf{U}$ is invertible, $\mathbf{A}^{(1)}$ is also a basis for M. Applying the above argument recursively, we get a sequence of matrices

$$\mathbf{A}^{(0)} = \mathbf{A}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(i)}, \ldots \tag{3.3}$$

such that $\mathbf{A}^{(i+1)} = \mathbf{A}^{(i)}\mathbf{U}^{(i)}$, where $\mathbf{U}^{(i)}$ is a transvection, and such that for each i there exist a column index j such that $\mathbf{A}_j^{(i)} >_{\mathbf{w}} \mathbf{A}_j^{(i+1)}$. As long as the matrix $\mathbf{A}^{(i)}$ is not reduced, the sequence continues. On the other hand, since $<_{\mathbf{w}}$ is a term order, and since at each step, the order of some column decreases, the sequence can not continue indefinitely. Thus there exists an n such that $\mathbf{A}^{(n)}$ is reduced. Furthermore, by definition of the transvections $\mathbf{U}^{(i)}$ it holds that

$$\deg_{\mathbf{w}}(\mathbf{A}) \geq \deg_{\mathbf{w}}\left(\mathbf{A}^{(1)}\right) \geq \deg_{\mathbf{w}}\left(\mathbf{A}^{(2)}\right) \geq \cdots \geq \deg_{\mathbf{w}}\left(\mathbf{A}^{(n)}\right).$$

Finally, since $\mathbf{A}^{(n)}$ is reduced it follows from the previous that

$$\deg_{\mathbf{w}}(\mathbf{A}) \geq \deg_{\mathbf{w}}\left(\mathbf{A}^{(n)}\right) \geq \deg_{\mathbf{w}}(\mathbf{A}'),$$

as desired. □

3.1.3 Objectives of the short-basis algorithm

The construction of the sequence in Equation (3.3) is exactly what is done in Buchberger's algorithm (with interreduction) for computing Gröbner bases of modules [16, p. 216]. In the algorithm, this process is continued until no more leading terms can be cancelled, i.e. until the basis matrix is reduced, and therefore the last half of the proof of Proposition 3.4 could have been done by referring to the fact that Buchberger's algorithm is always guaranteed to terminate. However, we have chosen to give the more elaborate proof as it almost also proves the following.

Proposition 3.5. *Let* $\mathbf{A}$ *be a basis of a module* $M \subseteq \mathbb{F}_q[x_1]^m$*. Then there exists an* $m \times m$ *invertible matrix*[1] $\mathbf{U}$ *with entries in* $\mathbb{F}_q[x_1]$ *such that either* $\deg_{\mathbf{w}}(\mathbf{AU}) < \deg_{\mathbf{w}}(\mathbf{A})$ *or* $\mathbf{AU}$ *is reduced with respect to* $\deg_{\mathbf{w}}$*.*

Proof. Let $\mathbf{A}^{(0)} = \mathbf{A}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(n)}$ be the sequence of matrices from the proof of Proposition 3.4. First, assume that $\deg_{\mathbf{w}}\left(\mathbf{A}^{(j+1)}\right) < \deg_{\mathbf{w}}\left(\mathbf{A}^{(j)}\right)$ for some j, then if we take $\mathbf{U} = \prod_{i=1}^{j} \mathbf{U}^{(i)}$, it holds that $\mathbf{AU} = \mathbf{A}^{(j+1)}$, and hence

$$\deg_{\mathbf{w}}(\mathbf{AU}) = \deg_{\mathbf{w}}\left(\mathbf{A}^{(j+1)}\right) < \deg_{\mathbf{w}}(\mathbf{A}).$$

Next, assume that for all j we have $\deg_{\mathbf{w}}\left(\mathbf{A}^{(j+1)}\right) = \deg_{\mathbf{w}}\left(\mathbf{A}^{(j)}\right)$, then if we take $\mathbf{U} = \prod_{i=1}^{n-1} \mathbf{U}^{(i)}$, it holds that $\mathbf{AU} = \mathbf{A}^{(n)}$, is reduced with respect to $\deg_{\mathbf{w}}$. In both of the above cases $\mathbf{U}$ is a product of transvections, and hence it is invertible. This proves the proposition. $\square$

Although the above proposition is formulated as an existence statement, its proof shows that the promised $\mathbf{U}$ can be computed as a composition of column operations (transvections). In Proposition 3.13 we shall have more to say about *how many* column operations are needed in this composition, but currently our main interest in Proposition 3.5 is that it proves that the following definition is not void.

Definition 3.6 ($\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$)**.** Let $\mathbf{A}$ be a basis of a module $M \subseteq \mathbb{F}_q[x_1]^m$. For a non-negative integer t, we let $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$ be an invertible matrix such that either

$$\deg_{\mathbf{w}}(\mathbf{A} \cdot \mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)) \leq \deg_{\mathbf{w}}(\mathbf{A}) - t,$$

or $\mathbf{A} \cdot \mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$ is reduced with respect to $\deg_{\mathbf{w}}$.

3.1.4 The short-basis algorithm

In this section we will use the developments in the previous sections to describe an efficient algorithm, along the lines of [2], for computing $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$. It is this algorithm we call the *short-basis algorithm.* The main idea of the algorithm is to divide the task of computing $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$ into smaller, and easier, problems. The key to this is the following observation. If t' is an integer such that $0 \leq t' \leq t$, then it holds that

$$\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t) = \mathbf{U}_{\mathbf{w}}(\mathbf{A}, t') \cdot \mathbf{U}_{\mathbf{w}}\left(\mathbf{A} \cdot \mathbf{U}_{\mathbf{w}}(\mathbf{A}, t'), t - d\right), \tag{3.4}$$

[1]We stress that $\mathbf{U}$ is invertible in the ring $\mathrm{Mat}_{m \times m}(\mathbb{F}_q[x_1])$, i.e. that its entries are polynomials.

where $d = \deg_{\mathbf{w}}(\mathbf{A}) - \deg_{\mathbf{w}}(\mathbf{A} \cdot \mathbf{U}_{\mathbf{w}}(\mathbf{A}, t'))$. Thus $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$ can be computed recursively as the product of the matrices on the right hand side of this equation. As indicated in the proof of Proposition 3.5 the base case $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, 1)$ can be done by a series of column operations. In Algorithm 1 this procedure is formalized and stated as pseudo code. Furthermore, pseudo code for the recursive algorithm for $t > 1$ implied by Equation (3.4) is given in Algorithm 2. Besides the computations directly suggested by Equation (3.4) an extra line (line 1) has been added. This line is crucial for the complexity of the algorithm, as the next section will show.

Algorithm 1 Algorithm for computing $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, 1)$

```
Input: An $m \times m$ matrix $\mathbf{A}$ with entries in $\mathbb{F}_q[x_1]$.
Input: A weight vector $\mathbf{w} \in \mathbb{N} \times \mathbb{Z}^m$.
Output: $\mathbf{U}(\mathbf{A}, 1)$
 1: $\mathbf{U} \leftarrow \mathbf{I}_m$
 2: repeat
 3:   if $\mathbf{A}$ is reduced with respect to $\deg_{\mathbf{w}}$ then
 4:     return U
 5:   else
 6:     Find $j_1 \neq j_2$ such that $i = \mathrm{LC}_{\mathbf{w}}(\mathbf{A}_{j_1}) = \mathrm{LC}_{\mathbf{w}}(\mathbf{A}_{j_2})$
 7:     if $\deg(\mathbf{A}_{i,j_1}) < \deg(\mathbf{A}_{i,j_2})$ then
 8:       Swap $j_1$ and $j_2$
 9:     end if
10:     $f \leftarrow -\frac{\mathrm{LT}(\mathbf{A}_{i,j_1})}{\mathrm{LT}(\mathbf{A}_{i,j_2})}$ // Leading terms with respect to ordinary degree
11:     $D_{old} \leftarrow \deg_{\mathbf{w}}(\mathbf{A}_{j_1})$
12:     $\mathbf{A}_{j_1} \leftarrow \mathbf{A}_{j_1} + f\mathbf{A}_{j_2}$
13:     $\mathbf{U}_{j_1} \leftarrow \mathbf{U}_{j_1} + f\mathbf{U}_{j_2}$
14:     $D_{new} \leftarrow \deg_{\mathbf{w}}(\mathbf{A}_{j_1})$
15:   end if
16: until $D_{new} < D_{old}$
17: return U
```

Remark 3.7. From Definition 3.6 it follows that $\mathbf{A} \cdot \mathbf{U}_{\mathbf{w}}(\mathbf{A}, \deg_{\mathbf{w}}(\mathbf{A}))$ must be reduced with respect to $\deg_{\mathbf{w}}$. Hence by Proposition 3.3 this matrix is also a Gröbner basis with respect to $<_{\mathbf{w}}$ for the module spanned by the columns of $\mathbf{A}$. This means that the short-basis algorithm can be used to compute Gröbner bases of modules over a univariate polynomial ring with respect to the weighted orders from Definition 2.23.

Example 3.8. In this example we make an attempt to visualize some of the notions introduced in the previous sections. We do so by illustrating a single

Algorithm 2 The short-basis algorithm for computing $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$

```
Input: An $m \times m$ matrix $\mathbf{A}$ with entries in $\mathbb{F}_q[x_1]$.
Input: A weight vector $\mathbf{w} \in \mathbb{N} \times \mathbb{Z}^m$.
Input: Integer $t$ such that $t \geq 1$.
Output: $\mathbf{U}(\mathbf{A}, t)$
1: $\mathbf{A} \leftarrow \pi_t(\mathbf{A})$
2: if $t = 1$ then
3:     return $\mathbf{U}(\mathbf{A}, 1)$ // Use Algorithm 1
4: else
5:     $\mathbf{U}' \leftarrow \mathbf{U}(\mathbf{A}, \lfloor t/2 \rfloor)$ // Recursive call
6:     $\mathbf{A}' \leftarrow \mathbf{A} \cdot \mathbf{U}'$
7:     return $\mathbf{U}' \cdot \mathbf{U}(\mathbf{A}', t - (\deg_{\mathbf{w}}(\mathbf{A}) - \deg_{\mathbf{w}}(\mathbf{A}')))$ // Recursive call
8: end if
```

iteration of the loop in Algorithm 1. As input matrix $\mathbf{A}$ we take the matrix with entries in $\mathbb{F}_4[x_1]$ defined as follows

$$\mathbf{A} = \begin{bmatrix} \alpha x_1^3 + x_1^2 + x_1 + x_1 & x_1^2 + \alpha & \alpha^2 x_1 + 1 & x_1 \\ x_1^2 + \alpha^2 & \alpha x_1 & x_1 + 1 & 1 \\ \alpha^2 x_1 + \alpha^2 & 1 & 1 & \alpha \\ \alpha^2 & 1 & \alpha & 1 \end{bmatrix}, \tag{3.5}$$

where α denotes a primitive element of $\mathbb{F}_4$. Furthermore, as weight vector we take

$$\mathbf{w} = (w_0, w_1, w_2, w_3, w_4) = (1, 0, 0, 0, 0).$$

In this case the weighted degree of a column $\mathbf{A}_j$ is

$$\deg_{\mathbf{w}}(\mathbf{A}_j) = \max_{i=1,\dots,4} \{\deg(\mathbf{A}_{i,j})\},$$

i.e. it is simply the maximal degree of an entry in $\mathbf{A}_j$. In the left hand part of Figure 3.1 the weighted degrees of the entries in $\mathbf{A}$ are visualized, by representing each entry $\mathbf{A}_{i,j}$ by a stack of $\deg(\mathbf{A}_{i,j}) + 1$ "cubes" (one for each coefficient). For instance the polynomial in $\mathbf{A}_{1,1}$ is represented by a stack of 4 cubes. Using this visualization, we have that the critical positions of a column simply are the positions that have the highest stacks. Furthermore, the leading coordinate is the position with the highest stack that is closest to the bottom left hand of the figure. For instance we see that the columns $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_4$ only have one critical position, whereas $\mathbf{A}_3$ has two.

The leading coordinate of each of the columns $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_4$ is one. Now, consider the loop in Algorithm 1 with $\mathbf{A}$ and $\mathbf{w}$ as inputs. Say that in line 6

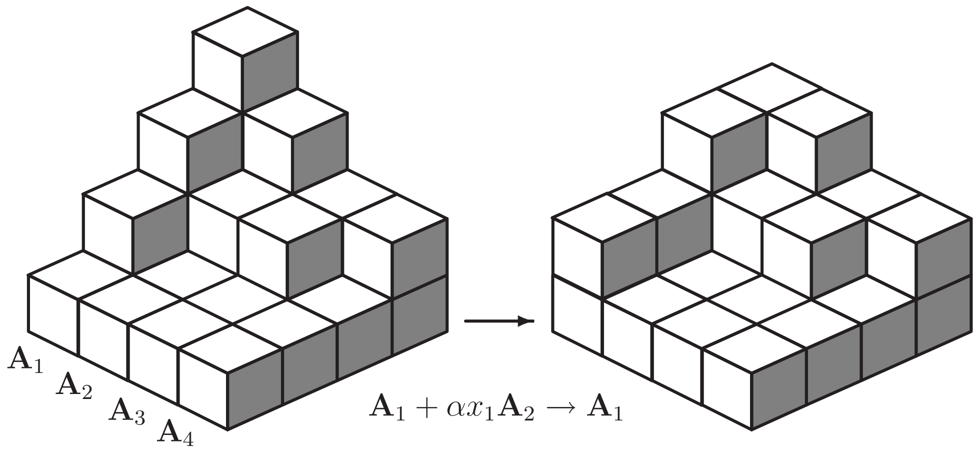


Figure 3.1: Visualization of the matrix $\mathbf{A}$ from Equation (3.5).

the algorithm finds $j_1 = 1$ and $j_2 = 2$ (and hence $i = 1$). It then computes

$$f = -\frac{\mathrm{LT}\left(\mathbf{A}_{1,1}\right)}{\mathrm{LT}\left(\mathbf{A}_{1,2}\right)} = \frac{\alpha x_1^3}{x_1^2} = \alpha x_1,$$

and in line 12 it makes the column operation $\mathbf{A}_1 \leftarrow \mathbf{A}_1 + \alpha x_1 \mathbf{A}_2$. The resulting matrix is illustrated in the right hand part of Figure 3.1. As expected we see that the leading term, with respect to $<_{\mathbf{w}}$, of $\mathbf{A}_1$ has been cancelled, and furthermore we see that the column operations has 'cut down' the maximal height of any stack in $\mathbf{A}_1$. This means that the weighted degree of $\mathbf{A}_1$ has been reduced by 1, and therefore Algorithm 1 terminates after the test in line 16. By applying Algorithm 1 again, the matrix can be further cut down. Ultimately this 'cutting down' process will yield a reduced matrix, as we saw in the proof of Proposition 3.4. We believe that thinking of the operations performed by the short-basis algorithm as repeatedly cutting down an $m \times m$ stack of cubes, is a helpful visualization that clarifies the notions introduced in the previous sections, as well as the analysis of the algorithm, which we will undertake next.

3.2 Analysis of the short-basis algorithm

In this section we estimate the complexity of the short-basis algorithm. The analysis will not depend on the specific weight vector $\mathbf{w}$ used in the weighted degree $\deg_{\mathbf{w}}$ and in the term order $<_{\mathbf{w}}$. Thus in the following we will take $\mathbf{w}$

to be some arbitrary but fixed weight vector in $\mathbb{N} \times \mathbb{Z}^m$, and to ease notation we will abbreviate $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$ to $\mathbf{U}(\mathbf{A}, t)$.

3.2.1 Sub-problems are easy

The problem with using Equation (3.4) directly for computing $\mathbf{U}(\mathbf{A}, t)$, is that the splitting of the problem does not in itself make the total problem easier, as the equation in words simply says: To reduce the weighted degree of $\mathbf{A}$ by t, first attempt to reduce it by t', and then try to reduce it further by t minus the amount by which the first attempt reduced the weighted degree of $\mathbf{A}$. The idea is that the sub-problems are actually easier to solve since one can use a simpler "approximation" of $\mathbf{A}$ in the place of $\mathbf{A}$ when solving them. It is this approximation π_t that is added in line 1 of Algorithm 2. We will now make precise what π_t does.

Definition 3.9 (Accuracy t approximation)**.** For a matrix $\mathbf{A}$ and an integer t, we define the *accuracy t approximation* of $\mathbf{A}$ to be the matrix $\pi_t(\mathbf{A})$ obtained as follows: If the (i, j)-th entry of $\mathbf{A}$ is $\sum_k a_k x_1^k$ then the (i, j)-th entry of $\pi_t(\mathbf{A})$ is

$$[\pi_t(\mathbf{A})]_{i,j} = \sum_{k \,:\, w_0 k + w_i > \deg_{\mathbf{w}}(\mathbf{A}_j) - t} a_k x_1^k.$$

Two matrices $\mathbf{A}$ and $\mathbf{B}$ are said to agree to precision t if $\pi_t(\mathbf{A}) = \pi_t(\mathbf{B})$, and this will be written as $\mathbf{A} \overset{t}{\sim} \mathbf{B}$.

Note that by definition of the accuracy t approximation it holds that $\mathbf{A} \overset{t}{\sim} \pi_t(\mathbf{A})$. The following lemma shows that we can use $\pi_t(\mathbf{A})$ in the place of $\mathbf{A}$ when computing $\mathbf{U}(\mathbf{A}, t)$. Thus, together with Equation (3.4), it proves the correctness of Algorithm 2. The lemma is a generalization of [2, Lemma 2.7].

Lemma 3.10. *It holds that*

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\pi_t(\mathbf{A}), t).$$

Proof. We will prove the lemma by induction on t, and we start with the base case $t = 1$. Since $\mathbf{A} \overset{1}{\sim} \pi_1(\mathbf{A})$, all the critical positions of $\mathbf{A}$ and $\pi_1(\mathbf{A})$ are the same. Note that Algorithm 1 stops as soon as for some column $\mathbf{A}_j$ the leading terms in all the critical positions in column j have been canceled. Therefore, since only the leading terms of critical positions are used in the computations performed by Algorithm 1, we have that $\mathbf{U}(\mathbf{A}, 1) = \mathbf{U}(\pi_1(\mathbf{A}), 1)$ and the base case follows. Actually the above argument proves a little more, namely that for $t \geq 1$ it holds that

$$\mathbf{U}(\mathbf{A}, 1) = \mathbf{U}(\pi_t(\mathbf{A}), 1). \tag{3.6}$$

Now for the induction step assume that $t > 1$. Let $\mathbf{V}$ be a matrix representing one of the column operation made in line 12 of Algorithm 1, i.e. a matrix such that $\mathbf{A} \cdot \mathbf{V}$ is the matrix obtained by making the column operation $\mathbf{A}_{j_1} \leftarrow \mathbf{A}_{j_1} + f\mathbf{A}_{j_2}$. Let

$$a^{(h)}(x_1) = \sum_k a_k^{(h)} x_1^h,$$

denote the polynomial in the (i, j_h)-th entry of $\mathbf{A}$, and let $f = \alpha x_1^\beta$ where

$$\beta = w_0^{-1} \left(\deg_{\mathbf{w}}\left(\mathbf{A}_{j_1}\right) - \deg_{\mathbf{w}}\left(\mathbf{A}_{j_2}\right)\right).$$

If $d \geq 0$ denotes $\deg_{\mathbf{w}}(\mathbf{A}) - \deg_{\mathbf{w}}(\mathbf{AV})$ then it holds that

$$\begin{aligned}
[\pi_t(\mathbf{A}) \cdot \mathbf{V}]_{i,j_1} &= \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_1}\right) - t} a_k^{(1)} x_1^k + \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_2}\right) - t} f \cdot a_k^{(2)} x_1^k \\
&= \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_1}\right) - t} a_k^{(1)} x_1^k + \sum_{w_0 (k-\beta) + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_2}\right) - t} \alpha a_{k-\beta}^{(2)} x_1^k \\
&= \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_1}\right) - t} a_k^{(1)} x_1^k + \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_2}\right) + w_0\beta - t} \alpha a_{k-\beta}^{(2)} x_1^k \\
&= \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left(\mathbf{A}_{j_1}\right) - t} \left(a_k^{(1)} + \alpha \cdot a_{k-\beta}^{(2)}\right) x_1^k \\
&= \sum_{w_0 k + w_i > \deg_{\mathbf{w}}\left((\mathbf{A}\cdot\mathbf{V})_{j_1}\right) - (t-d)} \left(a_k^{(1)} + \alpha \cdot a_{k-\beta}^{(2)}\right) x_1^k \\
&= \left[\pi_{t-d}(\mathbf{A} \cdot \mathbf{V})\right]_{i,j_1}.
\end{aligned}$$

Furthermore for any $j \neq j_1$ the j-th column of $\mathbf{AV}$ is equal to the j-th column of $\mathbf{A}$, and therefore we get that

$$\pi_t(\mathbf{A})\mathbf{V} \overset{t-d}{\sim} \pi_{t-d}(\mathbf{A} \cdot \mathbf{V}). \tag{3.7}$$

From Algorithm 1 it follows that we can write

$$\mathbf{U}(\mathbf{A}, 1) = \prod_{i=1}^{K} \mathbf{V}^{(i)},$$

where each $\mathbf{V}^{(i)}$ is a column addition of the type considered above. Let d_j be

$$d_j = \deg_{\mathbf{w}}\left(\mathbf{A}\cdot\prod_{i=1}^{j-1}\mathbf{V}^{(i)}\right) - \deg_{\mathbf{w}}\left(\mathbf{A}\cdot\prod_{i=1}^{j}\mathbf{V}^{(i)}\right),$$

and let $d = \sum_{j=1}^{K} d_j$ be the total amount by which $\mathbf{U}(\mathbf{A},1)$ reduces the weighted degree of $\mathbf{A}$, that is $d = \deg_{\mathbf{w}}(\mathbf{A}) - \deg_{\mathbf{w}}(\mathbf{A}\cdot\mathbf{U}(\mathbf{A},1))$. Then using Equation (3.7) repeatedly we get that

$$\begin{aligned}\pi_t(\mathbf{A})\cdot\mathbf{U}(\mathbf{A},1) = \pi_t(\mathbf{A})\cdot\prod_{i=1}^{K}\mathbf{V}^{(i)} &\overset{t-d_1}{\sim} \pi_{t-d_1}(\mathbf{A}\cdot\mathbf{V}_1)\cdot\prod_{i=2}^{K}\mathbf{V}_i \overset{t-(d_1+d_2)}{\sim}\cdots\\ &\overset{t-\sum d_i}{\sim} \pi_{t-\sum d_i}(\mathbf{A}\cdot\prod_{i=1}^{K}\mathbf{V}_i) \overset{t-d}{\sim} \pi_{t-d}(\mathbf{A}\cdot\mathbf{U}(\mathbf{A},1)).\end{aligned} \tag{3.8}$$

Using Equation (3.4) with $t'=1$, the “extended” induction base in (3.6) and the induction hypothesis, in that order, we get the following three equalities

$$\begin{aligned}\mathbf{U}(\pi_t(\mathbf{A}),t) &= \mathbf{U}(\pi_t(\mathbf{A}),1)\cdot\mathbf{U}\left(\pi_t(\mathbf{A})\mathbf{U}(\pi_t(\mathbf{A}),1),t-d\right)\\ &= \mathbf{U}(\mathbf{A},1)\cdot\mathbf{U}(\pi_t(\mathbf{A})\mathbf{U}(\mathbf{A},1),t-d)\\ &= \mathbf{U}(\mathbf{A},1)\cdot\mathbf{U}(\pi_{t-d}(\pi_t(\mathbf{A})\mathbf{U}(\mathbf{A},1)),t-d).\end{aligned}$$

Now, using Equation (3.8), the induction hypothesis, and Equation (3.4) again, we can continue the above chain of equalities by,

$$\begin{aligned}\mathbf{U}(\pi_t(\mathbf{A}),t) &= \mathbf{U}(\mathbf{A},1)\cdot\mathbf{U}(\pi_{t-d}(\mathbf{A}\mathbf{U}(\mathbf{A},1)),t-d)\\ &= \mathbf{U}(\mathbf{A},1)\cdot\mathbf{U}(\mathbf{A}\mathbf{U}(\mathbf{A},1),t-d).\\ &= \mathbf{U}(\mathbf{A},t).\end{aligned}$$

This completes the induction step. □

3.2.2 Combining sub-problems is easy

Lemma 3.10 shows that one only needs to consider certain monomials in the entries of $\mathbf{A}$ when computing $\mathbf{U}(\mathbf{A},t)$. This has a consequence for the type of monomials that can occur in the matrix $\mathbf{U}(\mathbf{A},t)$, and we now state and prove this result. For this we will need the notion of the length of a polynomial.

Definition 3.11 (Length)**.** The *length* of a polynomial

$$a(x_1) = \sum_{i=l}^{u} a_i x_1^i, \text{ with } a_l \neq 0, a_u \neq 0$$

is the maximal number of non-zero coefficients that $a(x_1)$ can have, namely $u - l + 1$.

The following lemma is a generalization of [2, Lemma 2.8].

Lemma 3.12. *Let* $\mathbf{A}$ *be an* $m \times m$ *matrix of polynomials in* $\mathbb{F}_q[x_1]$*. Any monomial in* $[\mathbf{U}(\mathbf{A}, t)]_{i,j}$ *is of the form* $\alpha x_1^{d_{ij}+e}$*, where* $|e| \leq w_0^{-1}(t-1)$ *and*

$$d_{ij} = w_0^{-1} \left(\deg_{\mathbf{w}} (\mathbf{A}_j) - \deg_{\mathbf{w}} (\mathbf{A}_i) \right).$$

In particular any polynomial in $\mathbf{U}(\mathbf{A}, t)$ *has length at most* $2tw_0^{-1}$*.*

Proof. We prove the lemma by induction on t, and we start with the base case $t = 1$. By Algorithm 1 we may factor $\mathbf{U}(\mathbf{A}, 1)$ as

$$\mathbf{U}(\mathbf{A}, 1) = \prod_{h=1}^{K} \mathbf{V}^{(h)}, \tag{3.9}$$

where each $\mathbf{V}^{(h)}$ is a matrix representing one of the column operations performed in line 13 of Algorithm 1. By line 10 of Algorithm 1 a monomial in the (j_2, j_1)-th position of $\mathbf{V}^{(h)}$ is of the form

$$f = -\frac{\mathrm{LT}\left(\tilde{\mathbf{A}}_{i,j_1}\right)}{\mathrm{LT}\left(\tilde{\mathbf{A}}_{i,j_2}\right)}, \tag{3.10}$$

where i is the common leading coordinate of column j_1 and j_2 and $\tilde{\mathbf{A}}$ is the matrix obtained by applying all the column operations $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(h-1)}$ (from the previous executions of the main loop) to $\mathbf{A}$. Note that Algorithm 1 stops as soon as the weighted degree of some column decreases, and hence for any index j we have that $\deg_{\mathbf{w}} (\mathbf{A}_j)$ remains unchanged during all executions of the algorithm's main loop. In particular for any of the values that (i, j_1, j_2) assumes during the execution it holds that

$$\begin{aligned} \deg\left(\mathrm{LT}\left(\tilde{\mathbf{A}}_{i,j_1}\right)\right) &= w_0^{-1} \left(\deg_{\mathbf{w}} (\mathbf{A}_{j_1}) - w_i \right), \\ \deg\left(\mathrm{LT}\left(\tilde{\mathbf{A}}_{i,j_2}\right)\right) &= w_0^{-1} \left(\deg_{\mathbf{w}} (\mathbf{A}_{j_2}) - w_i \right). \end{aligned}$$

Therefore, by Equation (3.10) it holds for any h, that the monomial in the (j_2, j_1)-th entry of $\mathbf{V}^{(h)}$ has degree

$$\deg\left(\mathrm{LT}\left(\tilde{\mathbf{A}}_{i,j_1}\right)\right) - \deg\left(\mathrm{LT}\left(\tilde{\mathbf{A}}_{i,j_2}\right)\right) = w_0^{-1} \left(\deg_{\mathbf{w}} (\mathbf{A}_{j_1}) - \deg_{\mathbf{w}} (\mathbf{A}_{j_2}) \right). \tag{3.11}$$

By (3.9) the (j_2, j_1)-th entry in $\mathbf{U}(\mathbf{A}, 1)$ is of the form

$$\sum_{\ell_1=1}^{m}\sum_{\ell_2=1}^{m}\cdots\sum_{\ell_{K-1}=1}^{m}[\mathbf{V}^{(1)}]_{j_2,\ell_1}\cdot[\mathbf{V}^{(2)}]_{\ell_1,\ell_2}\cdots[\mathbf{V}^{(K)}]_{\ell_{K-1},j_1}. \tag{3.12}$$

Now, by (3.11) the $(\ell_1, \ell_2, \ldots, \ell_{K-1})$-th summand in this expression is a monomial of degree

$$\begin{aligned}&\deg\left([\mathbf{V}^{(1)}]_{j_2,\ell_1}\cdot[\mathbf{V}^{(2)}]_{\ell_1,\ell_2}\cdots[\mathbf{V}^{(K)}]_{\ell_{K-1},j_1}\right) = \\ &\deg\left([\mathbf{V}^{(1)}]_{j_2,\ell_1}\right)+\sum_{h=1}^{K-2}\deg\left([\mathbf{V}^{(h+1)}]_{\ell_h,\ell_{h+1}}\right)+\deg\left([\mathbf{V}^{(K)}]_{\ell_{K-1},j_1}\right) = \\ &w_0^{-1}\left(\deg_{\mathbf{w}}(\mathbf{A}_{j_1})-\deg_{\mathbf{w}}(\mathbf{A}_{j_2})\right),\end{aligned}$$

where the last equality follows since the sum telescopes. Therefore the sum in (3.12) is also a monomial of this degree, and the induction base follows.

For the induction step assume that $t > 1$ and let $\mathbf{A}'$ denote $\mathbf{AU}(\mathbf{A}, 1)$. If we let

$$d = \deg_{\mathbf{w}}(\mathbf{A}) - \deg_{\mathbf{w}}(\mathbf{A}') \tag{3.13}$$

then by (3.4) we have that

$$\mathbf{U}(\mathbf{A}, t) = \mathbf{U}(\mathbf{A}, 1)\cdot\mathbf{U}(\mathbf{A}', t-d).$$

Therefore, by the induction hypothesis, any monomial in the (i, j)-th entry of $\mathbf{U}(\mathbf{A}, t)$ is of the form

$$\alpha_1 x_1^{d_{ik}}\cdot\alpha_2 x_1^{d'_{kj}+e} = \alpha_1\alpha_2 x_1^{d_{ik}+d'_{kj}+e}, \tag{3.14}$$

for some k, where $|e| \le w_0^{-1}(t-d-1)$ and $d'_{ij} = w_0^{-1}\left(\deg_{\mathbf{w}}\left(\mathbf{A}'_j\right) - \deg_{\mathbf{w}}(\mathbf{A}'_i)\right)$. We can rewrite the exponent of this monomial as

$$d_{ik} + d'_{kj} + e = d_{ik} + d_{kj} + d'_{kj} - d_{kj} + e = d_{ij} + e', \tag{3.15}$$

where $e' = d'_{kj} - d_{kj} + e$. Algorithm 1 stops immediately after the weighted degree of some column decreases. Therefore, for all but one column it holds that $\deg_{\mathbf{w}}(\mathbf{A}_h)$ is equal to $\deg_{\mathbf{w}}(\mathbf{A}'_h)$. Hence

$$d'_{kj} - d_{kj} = w_0^{-1}\left(\deg_{\mathbf{w}}\left(\mathbf{A}'_j\right) - \deg_{\mathbf{w}}(\mathbf{A}'_k) - (\deg_{\mathbf{w}}(\mathbf{A}_j) - \deg_{\mathbf{w}}(\mathbf{A}_k))\right)$$

is equal to either $w_0^{-1}(\deg_{\mathbf{w}}\left(\mathbf{A}'_j\right) - \deg_{\mathbf{w}}(\mathbf{A}_j))$ or $w_0^{-1}(\deg_{\mathbf{w}}(\mathbf{A}_k) - \deg_{\mathbf{w}}(\mathbf{A}'_k))$, and thus it follows from (3.13) that $|d'_{kj} - d_{kj}| \le w_0^{-1}d$. Using this, we then get

$$\begin{aligned}|e'| = |d'_{kj} - d_{kj} + e| &\le |d'_{kj} - d_{kj}| + |e| \\ &\le w_0^{-1}d + w_0^{-1}(t-d-1) = w_0^{-1}(t-1).\end{aligned} \tag{3.16}$$

Using the equations (3.15) and (3.16) on Equation (3.14), the induction step follows. □

Lemma 3.12 shows that the matrices

$$\mathbf{U}(\mathbf{A}, t') \text{ and } \mathbf{U}\left(\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t'), t - \left(\deg_{\mathbf{w}}(\mathbf{A}) - \deg_{\mathbf{w}}(\mathbf{A} \cdot \mathbf{U}(\mathbf{A}, t'))\right)\right)$$

computed when solving the sub-problems in (3.4) have entries of "small" length. This means that multiplication of entries in these matrices is fast, and hence combining the results of sub-problems, by computing the matrix product in (3.4), is fast too.

3.2.3 Complexity of the short-basis algorithm

In this section we use Lemma 3.12 to estimate the complexity of the short-basis algorithm. We will assume that the time the algorithm spends on making multiplications in $\mathbb{F}_q$ will dominate the total complexity. Thus all operations other than $\mathbb{F}_q$–multiplications performed by the algorithm (including $\mathbb{F}_q$–additions) will not count in its complexity.

Let $\mathrm{M}(t)$ denote the complexity of multiplying two polynomials of degree at most t in $\mathbb{F}_q[x_1]$. If we use the the Schönhage–Strassen algorithm [20, Thm. 8.23] for polynomial multiplication, we can take $\mathrm{M}(t)$ to be

$$\mathrm{M}(t) = \mathcal{O}(t \log t \log\log t). \tag{3.17}$$

Note that this is in fact also the complexity of multiplying two polynomials of *length* at most t. Before we can get an estimate of the complexity of the short-basis algorithm, we need the following proposition.

Proposition 3.13. *The total complexity of t consecutive calls to Algorithm 1, made from Algorithm 2, is $\mathcal{O}(m^2(t+m))$.*

Proof. Without loss of generality we will estimate the complexity of the *first* t calls to Algorithm 1 from Algorithm 2. Let $\mathbf{U}^{(i)}$ denote the result of the i-th call, i.e., $\mathbf{U}^{(i)} = \mathbf{U}(\mathbf{A}^{(i)}, 1)$, where $\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)}\mathbf{U}^{(i-1)}$ and $\mathbf{A}^{(1)} = \mathbf{A}$. As in the proof of Lemma 3.10 we can write

$$\mathbf{U}^{(i)} = \prod_{j=1}^{K} \mathbf{V}^{(j)},$$

where each $\mathbf{V}^{(j)}$ represents a column operation. Since Algorithm 1 stops immediately after a column degree decreases we have that

$$\deg_{\mathbf{w}}\left(\mathbf{A}^{(i)}\right) = \deg_{\mathbf{w}}\left(\mathbf{A}^{(i)} \cdot \prod_{j=1}^{K-1} \mathbf{V}^{(j)}\right),$$

and therefore, by a slight modification of Equation (3.8), that

$$\pi_1(\mathbf{A}^{(i)}) \prod_{j=1}^{K-1} \mathbf{V}^{(j)} \overset{1}{\sim} \pi_1\left(\mathbf{A}^{(i)} \cdot \prod_{j=1}^{K-1} \mathbf{V}^{(j)}\right) \overset{1}{\sim} \mathbf{A}^{(i)} \cdot \prod_{j=1}^{K-1} \mathbf{V}^{(j)}. \tag{3.18}$$

The last column operation $\mathbf{V}^{(K)}$ is the one that makes the degree of some column drop, and it only affects this single column. Therefore it follows from (3.18) that all the critical positions of $\pi_1(\mathbf{A}^{(i-1)}) \prod_{j=1}^{K-1} \mathbf{V}^{(j)}$ and $\mathbf{A}^{(i)}$ are the same, except in the column whose degree has dropped.

If one of the column additions made by Algorithm 1 when computing $\mathbf{U}^{(i)} = \mathbf{U}(\mathbf{A}^{(i)}, 1)$ does not make the degree of a column decrease, then the quantity

$$\sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(i)}\right), \tag{3.19}$$

must decrease. Furthermore, if a column operation makes the column degree decrease, then the quantity in (3.19) increases by at most $m-1$. Therefore if Algorithm 1 makes d_i column additions when computing $\mathbf{U}^{(i)}$, it follows from the observations after Equation (3.18) that

$$\sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(i)}\right) \leq \sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(i-1)}\right) - (d_{i-1} - 1) + (m-1).$$

This implies that

$$\sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(t)}\right) \leq \sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(1)}\right) - \sum_{i=1}^{t-1}(d_i - 1) + (m-1)(t-1),$$

and since $d_t \leq \sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(t)}\right) - m$ we get

$$\sum_{i=1}^{t}(d_i - 1) \leq \sum_{j=1}^{m} \mathrm{LC}_{\mathbf{w}}\left(\mathbf{A}_j^{(1)}\right) + (m-1)(t-1) - m - 1 \leq m^2 + mt - 2m - t.$$

Hence we get that the total number of column additions made in the t calls to Algorithm 1 is

$$\sum_{i=1}^{t} d_i \leq (m^2 + mt - 2m - t) + t = \mathcal{O}\left(m(t+m)\right).$$

Finally note that when Algorithm 1 is being called from Algorithm 2, all entries in $\mathbf{A}$ are monomials. Therefore the complexity of one column addition is $\mathcal{O}\left(\mathrm{M}\left(1\right) m\right) = \mathcal{O}\left(m\right)$, and hence we get that the total complexity of the t calls is $\mathcal{O}\left(m^2(t+m)\right)$ as desired. □

Using the above proposition along with Lemma 3.10 and Lemma 3.12, we can now get an estimate of the complexity of Algorithm 2. The theorem is a generalization of [2, Lemma 2.10].

Theorem 3.14 (Main)**.** *Let* $\mathbf{A}$ *be an* $m \times m$ *matrix of polynomials in* $\mathbb{F}_q[x_1]$*. The matrix* $\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$ *can be computed by Algorithm 2 in time at most*

$$\mathcal{O}\left(m^3 M\left(w_0^{-1} t\right) \log t + m^2 t\right).$$

Proof. We begin by estimating the complexity of the calls to Algorithm 1 made by Algorithm 2. Each call to Algorithm 1 makes t decrease by at least 1, and therefore Algorithm 2 makes at most t such calls. Therefore, by Proposition 3.13, the total complexity of the calls to Algorithm 1 is

$$\mathcal{O}\left(m^2(t+m)\right).$$

Let $T(t)$ denote the complexity of computing $\mathbf{U}(\mathbf{A}, t)$ with Algorithm 2, without counting the complexity of the calls to Algorithm 1. The complexity of multiplying two $m \times m$ matrices of polynomials, where each entry has length at most k is $\mathcal{O}\left(m^3 \mathrm{M}\left(k\right)\right)$. Therefore, since by Lemma 3.12 the length of a polynomial in any of the entries of the two matrices in the product in line 6 of Algorithm 2 is $\mathcal{O}\left(w_0^{-1} t\right)$, we get that

$$T(t) \leq \mathcal{O}\left(m^3 \mathrm{M}\left(w_0^{-1} t\right)\right) + 2T(t/2).$$

This implies that $T(t) = \mathcal{O}\left(m^3 \mathrm{M}\left(w_0^{-1} t\right) \log t\right)$, from which it follows that the total complexity of Algorithm 2 is

$$\mathcal{O}\left(m^3 \mathrm{M}\left(w_0^{-1} t\right) \log t + m^2(t+m)\right) = \mathcal{O}\left(m^3 \mathrm{M}\left(w_0^{-1} t\right) \log t + m^2 t\right).$$

□

Remark 3.15. We remark that the matrix multiplications performed in Algorithm 2, when combining the results of the sub-problems, can theoretically be done slightly faster than claimed in the proof of Theorem 3.14. In [64] Strassen gave an algorithm for computing the product of two square matrices of dimension m using $\mathcal{O}\left(m^{2.8}\right)$ multiplications. If Strassen's multiplication algorithm is used in Algorithm 2, we therefore get, by the same proof as above, that its complexity is

$$\mathcal{O}\left(m^{2.8}\mathrm{M}\left(w_0^{-1}t\right)\log t + m^2(t+m)\right).$$

Chapter 4

Multivariate polynomial interpolation

Key-equations for interpolation problems over simple C_{ab} curves were derived in Section 2.4. In this chapter we will show that these equations can be applied, together with the short-basis algorithm from Chapter 3, to get an efficient multivariate interpolation algorithm capable of solving the interpolation problem posed in Problem 2.8.

4.1 Solving the interpolation problem

4.1.1 The interpolation algorithm

Let $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$ be an interpolation problem as defined in Problem 2.8. In this section we will describe how the short-basis algorithm, together with the key-equations in Proposition 2.25, can be used to solve this problem. The following observation will be our starting point.

Proposition 4.1. *Let* $\mathbf{w} \in \mathbb{N} \times \mathbb{Z}^m$ *be a weight vector, and let* $\mathbf{A}$ *be an* $m \times m$ *matrix with entries in* $\mathbb{F}_q[x_1]$*. Let* $M \subseteq \mathbb{F}_q[x_1]^m$ *be the* $\mathbb{F}_q[x_1]$*–module spanned by the columns* $\mathbf{A}_1, \ldots, \mathbf{A}_m$ *of* $\mathbf{A}$*, and assume that* $\mathbf{A}$ *is reduced with respect to* $\deg_{\mathbf{w}}$*. Then it holds that*

$$\min_{\mathbf{x} \in M \setminus \{\mathbf{0}\}} \{\deg_{\mathbf{w}}(\mathbf{x})\} = \min_{j=1,\ldots,m} \{\deg_{\mathbf{w}}(\mathbf{A}_j)\}.$$

Proof. Since $\mathbf{A}$ is reduced with respect to $\deg_{\mathbf{w}}$ is follows from Proposition 3.3 that $\mathbf{A}$ is also a Gröbner basis for M with respect to $<_{\mathbf{w}}$. Therefore for any vector $\mathbf{x} \in M \setminus \{\mathbf{0}\}$ it holds that $\mathbf{A}_j \leq_{\mathbf{w}} \mathbf{x}$. Since by definition the order $<_{\mathbf{w}}$ measures *term over position* this means that $\deg_{\mathbf{w}}(\mathbf{A}_j) \leq \deg_{\mathbf{w}}(\mathbf{x})$ for all $\mathbf{x} \in M \setminus \{\mathbf{0}\}$, and the proposition follows. □

In the following, the parameters $(\mathbf{P}, \mathbf{y}, s, \ell, \mathbf{w}, \Delta)$ of the interpolation problem will be fixed. In particular the polynomial E and the polynomial vector $\mathbf{R}$ from Theorem 2.12 will be understood, and for brevity we will write $\mathbf{B}$ instead of $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ for the matrix from Definition 2.21. We will let

$$\tilde{\mathbf{w}} = \rho(\mathbf{w}),$$

denote the "expanded" weight vector defined in Equation (2.22). From the key-equation formulation of the interpolation problem in Proposition 2.25, we know that solving the interpolation problem is equivalent to finding a vector in the $\mathbb{F}_q[x_1]$–column span of $\mathbf{B}$ of weighted degree (with respect to $\tilde{\mathbf{w}}$) strictly less than Δ. To solve this problem we can use the short-basis algorithm to compute

$$\mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1).$$

By definition of the short-basis algorithm, one of the following two things will happen.

1. The weighted degree of $\mathbf{B} \cdot \mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1)$ is less than

   $$\deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - (\deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1) = \Delta\gamma m_\ell - 1.$$

   Since there are γm_ℓ columns in $\mathbf{B}$ this means that at least one column must have weighted degree strictly less than Δ, and since this column is in the column span of $\mathbf{B}$ we can then use it to get a valid interpolation polynomial as explained in Proposition 2.25.

2. The matrix $\mathbf{B} \cdot \mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1)$ is reduced with respect to $\deg_{\tilde{\mathbf{w}}}$. In this case it may still happen that one of the columns has weighted degree strictly less than Δ, in which case we again obtain a valid interpolation polynomial. However, it can also be that all columns have a larger weighted degree and in this case Proposition 4.1 shows that no interpolation polynomial of weighted degree strictly less than Δ exists.

Thus knowing $\mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1)$ suffices to solve the interpolation problem: If a valid interpolation polynomial exists then we can read it off from

$$\mathbf{B} \cdot \mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1), \tag{4.1}$$

Algorithm 3 Algorithm for solving interpolation problems.

```
Input: Interpolation problem $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$.
Output: A valid interpolation polynomial for the interpolation problem, if
    one exists, and failure otherwise.
1: Let $\tilde{\mathbf{w}} = \rho(\mathbf{w})$.
2: Set up the matrix $\mathbf{B} = \mathbf{B}_{s,\ell}(\mathbf{R}, E)$ as described in Definition 2.21
3: Compute $\mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1)$ with the short-basis algorithm.
4: Compute $\tilde{\mathbf{B}} = \mathbf{B} \cdot \mathbf{U}_{\tilde{\mathbf{w}}}(\mathbf{B}, \deg_{\tilde{\mathbf{w}}}(\mathbf{B}) - \Delta\gamma m_\ell + 1)$.
5: if there exists a column $\mathbf{q} = (q_{i,\mathbf{j}})$ in $\tilde{\mathbf{B}}$ such that $\deg_{\tilde{\mathbf{w}}}(\mathbf{q}) < \Delta$ then
6:     return
           $$Q(\mathbf{T}) = \sum_{i=0}^{\gamma-1} \sum_{\mathbf{j} \in \Delta_\ell} q_{i,\mathbf{j}}(x_1) x_2^i \mathbf{T}^{\mathbf{j}},$$
7: else
8:     return failure
9: end if
```

and if no valid interpolation polynomial exists we can detect this fact by seeing whether the matrix in (4.1) is reduced with respect to $\deg_{\tilde{\mathbf{w}}}$. In Algorithm 3 these observations are turned into an algorithm for solving the interpolation problem.

4.1.2 Analysis of the interpolation algorithm

We now turn to analysing the complexity of Algorithm 3. As in Section 3.2 we will assume that the time the algorithm spends on making multiplications in $\mathbb{F}_q$ will dominate the total complexity. If a little care is taken when setting up the matrix in line 2, the efforts spent in line 3 will dominate the complexity of the algorithm. The details of the analysis are given in Appendix A, and we collect the results derived there in the following theorem.

Theorem 4.2. *Let $(\mathbf{P}, \mathbf{y}, s, \ell, \mathbf{w}, \Delta)$ be an interpolation problem, and let $\mathbf{R} \in \mathcal{R}^v$ be the vector of polynomials from Theorem 2.12. If $w_j \leq \deg_{\mathbf{w}}(R_j)$ for $1 \leq j \leq v$, then the complexity of Algorithm 3 is*

$$\mathcal{O}\left(\ell^{3v}\gamma^3 M\left(\ell^{v+1}\gamma(N+\delta)\right)\log(\ell^{v+1}\gamma(N+\delta))\right).$$

Proof. We break the analysis in three parts.

Line 2. By Lemma A.4 the complexity of setting up $\mathbf{B}$ is at most

$$\mathcal{O}\left(\ell^{2v}\gamma \mathrm{M}\left((\ell(\gamma\delta+N)+\gamma^2\delta)\gamma\right)\right).$$

Line 3. This is the main step of the algorithm. By assumption $w_j \leq \deg_{\mathbf{w}}(R_j)$ for $1 \leq j \leq v$, and therefore it follows from Lemma A.5 that $\deg_{\tilde{\mathbf{w}}}(\mathbf{B}) = \mathcal{O}\left(\ell^{v+1}\gamma^2(N+\delta)\right)$. Since the number of rows and columns of $\mathbf{B}$ is $\gamma m_\ell = \mathcal{O}\left(\gamma\ell^v\right)$ we therefore get from Theorem 3.14 that the complexity of line 3 is

$$\begin{aligned}\mathcal{O}\left(\ell^{3v}\gamma^3\mathrm{M}\left(\ell^{v+1}\gamma(N+\delta)\right)\log\left(\ell^{v+1}\gamma^2(N+\delta)\right)+\ell^{3v+1}\gamma^4(N+\delta)\right)\\ =\mathcal{O}\left(\ell^{3v}\gamma^3\mathrm{M}\left(\ell^{v+1}\gamma(N+\delta)\right)\log\left(\ell^{v+1}\gamma(N+\delta)\right)\right),\end{aligned}$$

where the equality follows by Equation (3.17).

Line 4-9 The complexity of these lines are dominated by the complexity of line 2 and 3.

Comparing the complexities of the individual steps, we see that the complexity of line 3 dominates the overall complexity of Algorithm 3, and the theorem follows. □

Remark 4.3. For the applications of Algorithm 3 in list decoding that we have in mind, the assumptions $w_j \leq \deg_{\mathbf{w}}(R_j)$ in Theorem 4.2 will always be satisfied, unless the interpolation problem is trivial (i.e. if the word to be list decoded is itself a codeword). Thus this assumption is a mild one. Furthermore, if for some j it holds that $w_j > \deg_{\mathbf{w}}(R_j)$ then we claim that the interpolation problem can be reduced to a simpler interpolation problem involving one less variable.

To see this, consider an interpolation problem $(\mathbf{P}, \mathbf{y}, s, \ell, \mathbf{w}, \Delta)$, and assume (for notational simplicity) that $w_v > \deg_{\mathbf{w}}(R_v)$. Let $Q(\mathbf{T})$ be a valid interpolation polynomial for this interpolation problem of *least* weighted degree. We can write

$$Q(\mathbf{T}) = (T_v - R_v)^h\tilde{Q}(\mathbf{T}), \tag{4.2}$$

where $\tilde{Q}(\mathbf{T})$ is not divisible by $T_v - R_v$, and $0 \leq h \leq \ell$. The polynomial

$$P(\mathbf{T}) = (T_v - R_v)^h\tilde{Q}(T_1, \ldots, T_{v-1}, R_v),$$

is non-zero and is also a valid interpolation polynomial. Furthermore, if $\tilde{Q}(\mathbf{T})$ is not an element of $\mathcal{R}[T_1, \ldots, T_{v-1}]$ then $\deg_{\mathbf{w}}(P) < \deg_{\mathbf{w}}(Q)$, which is a contradiction. Hence $\tilde{Q}(\mathbf{T})$ must be in $\mathcal{R}[T_1, \ldots, T_{v-1}]$. Now consider the following $\ell + 1$ "punctured" interpolation problems in $v - 1$ variables,

$$(\mathbf{P}', \mathbf{y}', s-h, \ell-h, \mathbf{w}', \Delta - hw_v), \quad \text{for } 0 \leq h \leq \ell \tag{4.3}$$

where $\mathbf{P}'$, $\mathbf{y}'$ and $\mathbf{w}'$ are obtained by dropping the v-th coordinate from $\mathbf{P}$, $\mathbf{y}$ and $\mathbf{w}$ respectively. The above discussion shows that to solve the interpolation problem $(\mathbf{P}, \mathbf{y}, s, \ell, \mathbf{w}, \Delta)$ it suffices to solve the problems in Equation (4.3). If $w_j \leq \deg_{\mathbf{w}}(R_j)$ for $1 \leq j \leq v-1$ then by Theorem 4.2, this can be done in complexity

$$\mathcal{O}\left(\ell \cdot \ell^{3v-3}\gamma^3 \mathrm{M}\left(\ell^v\gamma(N+\delta)\right)\log(\ell^v\gamma(N+\delta))\right) \\ = \mathcal{O}\left(\ell^{3v-2}\gamma^3 \mathrm{M}\left(\ell^v\gamma(N+\delta)\right)\log(\ell^v\gamma(N+\delta))\right),$$

which is smaller than the complexity of the original interpolation problem $(\mathbf{P}, \mathbf{y}, s, \ell, \mathbf{w}, \Delta)$ when $w_j \leq \deg_{\mathbf{w}}(R_j)$ for $1 \leq j \leq v$. If more coordinates j have $w_j > \deg_{\mathbf{w}}(R_j)$, we can apply the above argument recursively to these. Altogether this shows that interpolation problems where $w_j > \deg_{\mathbf{w}}(R_j)$ for some j, are easier than those without this property. Thus the assumptions in Theorem 4.2 are mild, and can be removed by preprocessing the interpolation problem as described above.

4.2 List decoding simple C_{ab} codes

In this section we will apply Algorithm 3 to solve the interpolation step of the G–S algorithm for list decoding algebraic-geometry codes defined over simple C_{ab} curves.

4.2.1 The interpolation step

We begin by defining the class of simple C_{ab} codes.

Definition 4.4 (Simple C_{ab} codes)**.** Let $\mathcal{C}$ be a simple C_{ab} curve over $\mathbb{F}_q$, and let $P_1, \ldots, P_n$ be distinct places satisfying Assumption 2.1. Let $D = \sum_{i=1}^n P_i$ and $G = \mu P_\infty$ for some $\mu \geq 0$. For these parameters, we define a *simple C_{ab} code* to be the algebraic-geometry code

$$C_{\mathcal{L}}(D, G) = \{(f(P_1), \ldots, f(P_n)) \mid f \in \mathcal{L}(G)\} \subseteq \mathbb{F}_q^n.$$

In the following we will let the simple C_{ab} curve $\mathcal{C}$ on which a simple C_{ab} code is defined, be fixed. In particular we let δ, γ and $\mathcal{R}$ be as defined in Section 2.3 for this fixed curve. By Lemma 2.18 the genus of $\mathcal{C}$ is $\frac{1}{2}(\delta-1)(\gamma-1)$, and therefore by the Goppa bound [63, II.2.3] we have the following.

Proposition 4.5. *Let $C_{\mathcal{L}}(D, G)$ be a simple C_{ab} code.*

- *The length of $C_{\mathcal{L}}(D, G)$ is $n = |D|$.*

- *The dimension of $C_{\mathcal{L}}(D,G)$ is $k = \dim(\mathcal{L}(G))$.*
- *The minimum distance of $C_{\mathcal{L}}(D,G)$ is at least $n-k+1-\frac{1}{2}(\delta-1)(\gamma-1)$.*

Let $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}_q^n$ be a received word of a simple C_{ab} code, and say that we wish to correct τ errors. To list decode $\mathbf{y}$ with the G–S algorithm [30], with *multiplicity parameter* s and *designed list size* ℓ, we need to find a non-zero polynomial $Q(T) \in \mathcal{R}[T]$ such that

1. The degree in T of $Q(T)$ is at most ℓ.
2. For $1 \leq i \leq n$, $Q(T)$ has a zero of multiplicity at least s in (P_i, y_i).
3. The weighted degree $\deg_\mu(Q(T))$ is strictly smaller than $s(n-\tau)$.

Let $Q(T)$ be a polynomial satisfying the above requirement, and let $f \in \mathcal{L}(G)$ be a polynomial generating a codeword agreeing with $\mathbf{y}$ in at least $n-\tau$ positions. Then by item 2 above, the degree of the zero-divisor of $Q(f)$ satisfies

$$\deg((Q(f))_0) \geq s(n-\tau). \tag{4.4}$$

Furthermore, by definition of the weighted degree, the degree of the pole-divisor of $Q(f)$ satisfies

$$\deg((Q(f))_\infty) \leq \deg_\mu(Q) < s(n-\tau), \tag{4.5}$$

By [63, I.4.11] the equations (4.4) and (4.5) can only hold simultaneously if $Q(f) = 0$. Thus codewords agreeing with $\mathbf{y}$ in at least $n-\tau$ positions can be found as roots in $Q(T)$. Computing such roots is the content of the *root-finding step* of the G–S algorithm. Note that $Q(T)$ can have at most ℓ roots f for which $Q(f) = 0$, and hence there can be at most ℓ codewords that are "close" to $\mathbf{y}$. This explains the name "designed list size" for ℓ used above. We now show that Algorithm 3 can be applied to efficiently solve the interpolation step in the G–S algorithm.

Proposition 4.6. *The interpolation step of the Guruswami–Sudan list decoding algorithm for simple C_{ab} codes, with designed list size ℓ, can be done in complexity*

$$\mathcal{O}\left(\ell^3\gamma^3 M\left(\ell^2\gamma(N+\delta)\right)\log(\ell\gamma(N+\delta))\right). \tag{4.6}$$

Proof. The requirements on the polynomial $Q(T)$ listed above correspond exactly to an instance of Problem 2.8 in $v = 1$ indeterminates, namely the interpolation problem

$$(\mathbf{P}, \mathbf{y}, s, \ell, \mu, s(n-\tau)),$$

where $\mathbf{P} = (P_1, \ldots, P_n)$. By Theorem 4.2 (and the remark following it) this interpolation problem can be solved in the complexity stated in Equation (4.6). $\square$

Remark 4.7. There exist efficient algorithms for the root-finding step of the G–S algorithm for algebraic-geometry codes [3, 69]. In practice the interpolation step is often the most computationally heavy part of the list decoder.

4.2.2 Examples

We now give two examples of simple C_{ab} codes that have been intensively studied in the literature, and we use Proposition 4.6 to estimate the complexity of the interpolation step in the G–S algorithm for these codes. We also compare the resulting complexities to those of other algorithms for the same task.

Example 4.8 (Reed–Solomon codes)**.** As we saw in Section 2.5, the projective line $\mathbb{P}^1_{\mathbb{F}_q}$ is a simple C_{ab} curve with $\delta = 0$ and $\gamma = 1$, and algebraic-geometry codes defined on the projective line, are Reed–Solomon codes. Let $P_1, \ldots, P_n$ be distinct places of $\mathbb{F}_q(\mathbb{P}^1_{\mathbb{F}_q}) = \mathbb{F}_q(x_1)$ then these places satisfy Assumption 2.1. Therefore if we let $D = \sum_{i=1}^n P_i$ and $G = (k-1)P_\infty$, we get that the algebraic-geometry code $C_{\mathcal{L}}(D, G)$ is a Reed–Solomon code of length n and dimension k, and furthermore that this is a simple C_{ab} code. By Proposition 4.6 this means that we can use Algorithm 3 to solve the interpolation step of the G–S algorithm for Reed–Solomon codes, and since $n = \gamma N = N$ we get that this can be done in complexity

$$\mathcal{O}\left(\ell^3 \mathrm{M}\left(\ell^2 n\right) \log(\ell n)\right) = \mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right). \tag{4.7}$$

Actually the general analysis in Theorem 4.2 is slightly wasteful for the special case of Reed–Solomon codes. By Corollary 2.26 the key-equations for Reed–Solomon codes are such that the weighted degree of the matrix $\mathbf{B} = \mathbf{A}_\ell(-R)\mathbf{D}_{s,\ell}(E)$ used in Algorithm 3, is at most

$$\sum_{j=0}^{s-1}(sn - j) + \sum_{j=s}^{\ell} j(n-1) = \binom{\ell+1}{2}(n-1) + \binom{s+1}{2}n.$$

The amount by which the short-basis algorithm is required to reduce the weighted degree of $\mathbf{B}$ in line 3 of Algorithm 3 is

$$t = \deg_{\mathbf{w}}(\mathbf{B}) - (\ell+1)s(n-\tau).$$

The parameters s and ℓ in the G–S algorithm for Reed–Solomon codes are chosen such that they satisfy

$$\binom{s+1}{2} n < (\ell+1)s(n-\tau) - \binom{\ell+1}{2}(k-1),$$

see Proposition 5.5 or [30] for a proof of this fact. Therefore we have

$$t = \binom{\ell+1}{2}(n-1) + \binom{s+1}{2} n - (\ell+1)s(n-\tau) < \binom{\ell+1}{2}(n-k).$$

If we use this estimate in the proof of Theorem 4.2, then we get that the interpolation step for Reed–Solomon codes can be done by Algorithm 3 in complexity

$$\mathcal{O}\left(\ell^3 \mathrm{M}\left(\ell^2(n-k)\right)\log(\ell(n-k)) + \ell^2 \mathrm{M}\left(\ell n\right)\right).$$

The first term in this expression is the complexity of the computation made by the short-basis algorithm, and the second term is the complexity of setting up the matrix $\mathbf{B}$. From the above improved analysis we see that the complexity of the short-basis algorithm's computations actually depends on $n-k$ rather than on n. This behaviour is similar to what can be achieved with the re-encoding technique described in Section 2.5.4. Thus we conclude that the benefits that can be gained by re-encoding are in a sense already 'built into' the short-basis algorithm.

We now compare the complexity estimate in Equation (4.7), to that of other algorithms in the literature. The interpolation step of the G–S algorithm for Reed–Solomon codes may be formulated and solved as a linear algebra problem [30], in which case its complexity is $\mathcal{O}(\ell^6 n^3)$. In [49] Olshevsky and Shokrollahi used the fact that the linear algebra problem has low so-called displacement rank, to reduce the complexity of the interpolation step to $\mathcal{O}(\ell^5 n^2)$. The same complexity was achieved by Lee and O'Sullivan by a Gröbner basis based technique [43], and by Sakata, Numakami and Fujisawa [58] using a generalization of the Berlekamp–Massey algorithm from [56]. The short-basis algorithm described in Section 3.1 is a generalization of an algorithm by Alekhnovich from [2]. In the same paper Alekhnovich also showed how this algorithm can be applied to solve the interpolation step of the G–S algorithm for Reed–Solomon codes, resulting in an algorithm with complexity $\mathcal{O}\left(\ell^8 n \log^2(\ell n) \log\log(\ell n)\right)$. This was the first algorithm for the interpolation step with a complexity that is less than quadratic in the code length.

Comparing the complexities of the above algorithms, to the complexity in Equation (4.7) we see that asymptotically our algorithm compares favourably to these. However, as we will see in Section 4.2.3 it is not straight forward to make an implementation of the short-basis algorithm that puts this theoretical advantage into practice. Therefore the algorithms mentioned above may be faster than Algorithm 3 for short codes.

Example 4.9 (Hermitian code)**.** This is a continuation of Example 2.20, where it was shown that the Hermitian curve $\mathcal{H}$ over $\mathbb{F}_q = \mathbb{F}_{r^2}$ defined by the equation

$$x_2^r + x_2 = x_1^{r+1},$$

is a simple C_{ab} curve with $\delta = r + 1$ and $\gamma = r$. Furthermore, it was shown that there are $n = r^3$ places $P_1, \ldots, P_n$ in $\mathbb{F}_q(\mathcal{H})$ satisfying Assumption 2.1. The *Hermitian code* is the algebraic-geometry code $C_{\mathcal{L}}(D, G)$ of length r^3 and alphabet $\mathbb{F}_{r^2}$, obtained by letting $D = \sum_{i=1}^n P_i$ and $G = \mu P_\infty$ for some $\mu \geq 0$. We see that the Hermitian code satisfies the requirements to be a simple C_{ab} code, and in particular it follows from Proposition 4.6 that the interpolation step in the G–S algorithm for Hermitian codes, can be done in complexity

$$\begin{aligned}\mathcal{O}\left(\ell^3 r^3 \mathrm{M}\left(\ell^2 r^3\right) \log(\ell r^3)\right) &= \mathcal{O}\left(\ell^3 n \mathrm{M}\left(\ell^2 n\right) \log(\ell n)\right) \\ &= \mathcal{O}\left(\ell^5 n^2 \log^2(\ell n) \log\log(\ell n)\right)\end{aligned}$$

In [49] Olshevsky and Shokrollahi gave an algorithm for the interpolation step of the Sudan list decoding algorithm (i.e. for the interpolation problem without multiplicities) for Hermitian codes, using the notion of displacement rank. The complexity of their algorithm is $\mathcal{O}\left(\ell n^{7/3}\right)$. In [57] Sakata showed how a generalization of the Berlekamp–Massey algorithm from [56], can be used in the interpolation step of the G–S algorithm for general one-point algebraic-geometry codes. When specialized to Hermitian codes, this yields an algorithm with complexity $\mathcal{O}\left(\ell^6 n^{8/3}\right)$. Recently Lee and O'Sullivan [41, 42] have used a specialized Gröbner basis algorithm to tackle the interpolation step in the G–S algorithm for Hermitian codes, resulting in an algorithm with complexity $\mathcal{O}\left(\ell^5 n^{8/3}\right)$. We conclude that our algorithm is asymptotically faster than other algorithms known in the literature. As before it should be noted that this asymptotic advantage may however only take effect for long codes.

Example 4.10 (General simple C_{ab} codes)**.** The algorithm in [57] handles the interpolation step in the G–S algorithm for general one-point algebraic-geometry codes. In our notation its complexity is

$$\mathcal{O}\left((\gamma+\delta)^2\ell^6 n^2\right).$$

For most simple C_{ab} codes the complexity of Algorithm 3 compares favourably to this.

Remark 4.11. Recently the paper [66] has reported promising results from experiments with a probabilistic algorithm for the interpolation step of the G–S algorithm for Reed–Solomon codes. Like Algorithm 3, this algorithm proceeds by divide–and–conquer. However, as opposed to Algorithm 3, the divide–and–conquer approach is applied to the multiplicity parameter s of the interpolation problem. This means that an interpolation problem with multiplicity parameter $s_1 + s_2$ is solved by combining the results of two interpolation problems with multiplicity parameters s_1 and s_2. In [66] it is demonstrated that the algorithm experimentally performs better than the Lee–O'Sullivan interpolation algorithm [43]. However no general bounds on the expected running time of the algorithm are given.

4.2.3 Simulations

In this section we supplement the asymptotic analysis of Algorithm 3 from Section 4.1.2, with an investigation of the algorithm's performance for practical problem sizes. We do so by simulating data transmission over a noisy channel using Reed–Solomon codes of various lengths, that are decoded with the G–S list decoder. When list decoding the simulated received words, we use and compare two different algorithms for the interpolation step, namely Lee and O'Sullivan's interpolation algorithm [43] and Algorithm 3.

We begin by describing the codes used in the simulations. We use eight different Reed–Solomon codes all with rate (approximately) $R = \frac{7}{10}$. These codes are defined as follows: for $i \in \{6, \dots, 13\}$ we let C_i be the Reed–Solomon code of length $n_i = 2^i - 1$ and dimension $k_i = \lceil Rn_i \rceil$, defined over the alphabet $\mathbb{F}_{q_i} = \mathbb{F}_{2^i}$. The parameters of these codes are stated in Table 4.1.

Next, we describe how the simulations are made. Let n and k denote the length and dimension of a Reed–Solomon code respectively. Furthermore, let s be a multiplicity parameter, Δ a weighted degree bound and $\ell = \lfloor \frac{\Delta-1}{k-1} \rfloor$ a

C_i	n_i	k_i	Δ_i	ℓ_i	τ_i
C_6	63	45	268	6	9
C_7	127	89	537	6	19
C_8	255	179	1081	6	38
C_9	511	358	2167	6	77
C_{10}	1023	717	4341	6	154
C_{11}	2047	1433	8683	6	310
C_{12}	4095	2867	17374	6	620
C_{13}	8191	5734	34752	6	1240

Table 4.1: Parameters of the Reed–Solomon codes $C_6, \dots, C_{13}$.

designed list size parameter. If these parameters satisfy

$$\binom{s+1}{2} n < (\ell+1)\Delta - \binom{\ell+1}{2}(k-1), \tag{4.8}$$

then there exists an interpolation polynomial for any received word $\mathbf{y} \in \mathbb{F}_q^n$ with multiplicity parameter s and of weighted degree strictly less than Δ. For a proof of this fact see [30] or Proposition 5.5. We will use multiplicity parameter $s = 5$ in the simulations. For each code C_i in Table 4.1 we have computed the least Δ_i for which Equation (4.8) guarantees the existence of an interpolation polynomial of weighted degree strictly less than Δ_i. Note that $\ell_i = \lfloor \frac{\Delta_i - 1}{k_i - 1} \rfloor = 6$ for all codes in the table. Furthermore, for each code we have computed the largest integer τ_i smaller than $n_i - \frac{\Delta_i}{s}$. As we saw in Section 4.2.1, the G–S algorithm for C_i correctly list decodes any received word corrupted by at most τ_i errors.

For each of the codes C_i in Table 4.1, we simulate the transmission of a codeword from C_i over a noisy channel as follows:

1. Choose a polynomial in $\mathbb{F}_q[x_1]$ of degree less than k_i at random, and encode it to a codeword $\mathbf{c} \in C_i$.
2. Generate a received word $\mathbf{y}$ by adding τ_i errors at random to $\mathbf{c}$.

As mentioned above, we decode the received word $\mathbf{y}$ with the G–S list decoder, using two different algorithms for the interpolation step. The first of these is the Gröbner basis based algorithm by Lee and O'Sullivan [43]. Below we will call this algorithm the *Lee–O'Sullivan algorithm.*

In Example 4.8 we saw that, the asymptotic complexity of this algorithm is $\mathcal{O}(\ell^5 n^2)$. A characteristic property of the Lee–O'Sullivan algorithm is that it proceeds solely by elementary operations on a matrix of univariate polynomials. This means that the constant hidden in the big-oh estimate of the algorithm's complexity is relatively small (see the comment [42, p. 11]), and hence its efficiency is present already for small problem instances. In other words, the Lee–O'Sullivan algorithm is an efficient method for the interpolation step for practical code lengths. This is the reason why we have chosen to compare the performance of Algorithm 3, against this specific algorithm. We have implemented the Lee–O'Sullivan algorithm in the computer algebra system `Magma` [13], and the source code can be obtained from [14]. Furthermore, we have simulated data transmission using the codes $C_6, \ldots, C_{13}$ as described above, and used the Lee–O'Sullivan algorithm in the interpolation step. We have recorded the running times of these interpolation steps, and plotted them against the code lengths in Figure 4.1 (the solid curve). Each plotted running time represents the average of 10 simulated decodings. We remark that the code is only a proof–of–concept implementation, following the pseudo code from [43] closely. Thus we have made no serious attempts to optimize the code. The same holds for our implementation of Algorithm 3 which we describe below. Furthermore, all simulations are carried out on the same computer, and thus the implementations should offer a fair basis for comparing the performance of the algorithms.

The second algorithm for the interpolation step we consider is Algorithm 3. As we saw in Example 4.8, the complexity of using this algorithm in the interpolation step is

$$\mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right). \tag{4.9}$$

Algorithm 3 derives its efficiency from that of the short-basis algorithm, which in turn relies on the efficiency of Schönhage–Strassen's fast polynomial multiplication algorithm [20, p. 235]. The advantage of this algorithm over naive multiplication and the classic Karatsuba's algorithm, is only present for polynomials of a certain length[1]. This means that the short-basis algorithm is *inefficient* when it is used to reduce the degree of a basis by a small amount. Therefore, to make Algorithm 3 realize the efficiency promised by the asymptotic estimate in Equation (4.9), we need to make the following modification of the short-basis algorithm:

- We introduce a *threshold* parameter, denoted λ.

[1]`Magma` implements the Schönhage–Strassen algorithm. In the current implementation it is estimated that this algorithm outperforms both naive multiplication and Karatsuba's algorithm for polynomials of degree larger than 128 [1].

- In the short-basis algorithm, we only use the recursive divide–and–conquer approach as long as the target reduction-parameter t is strictly larger than the threshold λ. When t is smaller than λ, we use the Gaussian elimination approach from Algorithm 1 to reduce the weighted degree of the basis matrix.

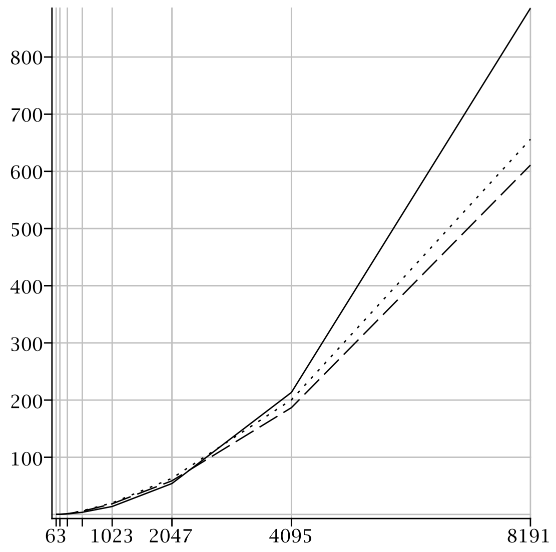
For simplicity we have avoided a formal description of the above modification of the short-basis algorithm. The description of the algorithm given in Section 3.1 corresponds to threshold $\lambda = 1$.

The modified short-basis algorithm avoids using the (slow) recursive approach on small problem instances. This is crucial for the practical performance of the algorithm. We have made a `Magma` implementation of Algorithm 3 (and hence also of the short-basis algorithm), and the source code can be obtained from [14]. We have simulated data transmission using the codes $C_6, \dots, C_{13}$ as described above, and used Algorithm 3 in the interpolation step. The running times of these interpolation steps are shown in Figure 4.1, for two different values of the threshold: one where $\lambda = 32$ (dotted curve) and one where $\lambda = 512$ (dashed curve). As above, each plotted running time represents the average of 10 simulated decodings. We have experimented with other values of the threshold parameter, and found that (almost) independent of the code length, the value $\lambda = 512$ gives the best performance.

We extract a number of conclusions from the graphs in Figure 4.1:

- Firstly we note that the Lee–O'Sullivan algorithm performs better than Algorithm 3 for codes of length less than approximately 3000. Another way to say this is that the *cross over point* between the algorithms, i.e. the code length at which they perform equally good, is about 3000. The observation that the Lee–O'Sullivan algorithm outperforms Algorithm 3 for small code lengths is expectable since, as commented above, the constant in the complexity estimate of the Lee–O'Sullivan algorithm is small.

- Secondly, we note that the cross over point between the algorithms, which the asymptotic analysis predicts *must* exist, actually occur within an observable running time. Thus we conclude that Algorithm 3 does offer some advantage over the more direct approach in the Lee–O'Sullivan algorithm. In particular we conclude that the algorithm is not purely theoretical, and that it realizes the performance promised by the asymptotic analysis for large, but not unreasonable, problem sizes.



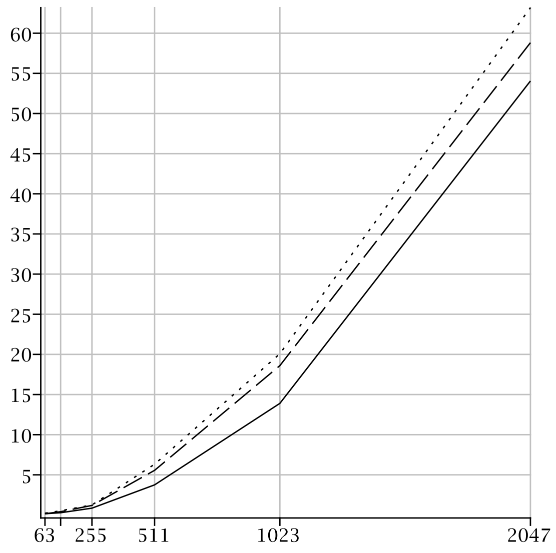


Figure 4.1: Plot of code length (first axis) against running time (in seconds) of three algorithms for the interpolation step. The curves show the running times of the Lee–O'Sullivan algorithm (solid) and of Algorithm 3 with threshold $\lambda = 32$ (dotted) and threshold $\lambda = 512$ (dashed). The top figure shows running times for code lengths between 63 and 8191, while the bottom figure is zoomed in on code lengths between 63 and 2047.

- Finally, we remark that even though the cross over point is not unreasonably large, it is probably still too large for many applications. Furthermore, for simulations with larger values of ℓ one can verify that the cross over point is even larger. This limits the practical applicability of Algorithm 3 somewhat. We have made a thorough investigation of what parts of Algorithm 3 are the most time consuming, and found that much time is used to compute the accuracy t approximation in the short-basis algorithm, and in general on data representation and memory management. Therefore we believe that if the short-basis algorithm is implemented in a more low-level programming language than `Magma` (e.g. in `C`), which allows direct access to data representations and memory, it will be possible to reduce the excess time spent on operations other than Gaussian elimination considerably. This in turn will lower the cross over point. We have not undertaken this low-level implementation task, as it is beyond the scope of this thesis work.

Chapter 5

Folded Reed–Solomon codes

In this chapter we give a detailed introduction to folded Reed–Solomon codes. We show how folded Reed–Solomon codes can be list decoded arbitrarily close to the optimal error-radius $1 - R$, and we describe the list decoder achieving this. The list decoder encompasses an interpolation step and a root-finding step. We show how the interpolation algorithm from Chapter 4 can be used to handle the interpolation step, and we present a method based on Hensel–lifting for solving the root-finding step. Furthermore, we compare these algorithms to others known in the literature.

The material in this chapter is a comprehensive extension of [7]. Furthermore, the results in Section 5.3, 5.4.3 and 5.5 are new.

5.1 Introduction

A Reed–Solomon code of rate R can be list decoded up to relative error-radius $1 - \sqrt{R}$ using the Guruswami–Sudan list decoding algorithm [30]. On the other hand we saw in Theorem 1.2 that no code of rate R can be list decoded beyond error-radius $1 - R$. A natural question is therefore whether one can construct explicit codes, and accompanying list decoders, with performances in the gap between the error-radii $1 - \sqrt{R}$ and $1 - R$. Building on work by Parvaresh and Vardy [50], Guruswami and Rudra [27–29] answered this question, and below we outline their result.

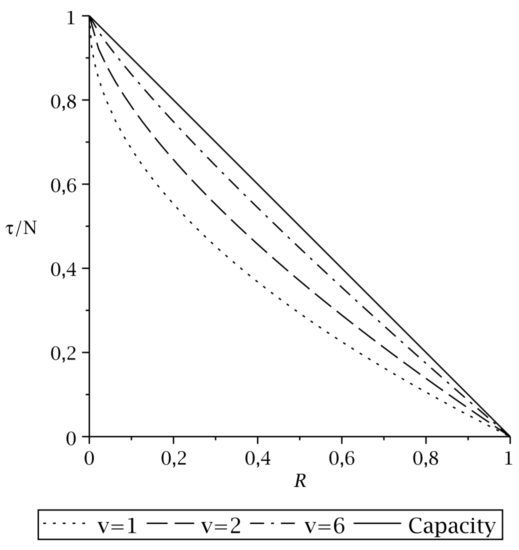


Figure 5.1: Plot of the error-radius $1 - R^{\frac{v}{v+1}}$ for various values of v.

5.1.1 Achieving capacity

Guruswami and Rudra showed that one can apply a *folding operation* to Reed–Solomon codes, to obtain codes that can be list decoded up to error-radius

$$1 - R^{\frac{v}{v+1}}, \tag{5.1}$$

where $v \geq 1$ is a parameter of the construction. Furthermore, the list decoders in the construction are completely explicit. As $v \to \infty$ the error-radius in (5.1) tends to $1-R$, and in this asymptotic sense the folded Reed–Solomon codes have optimal list decoders. In [27] this fact is expressed by saying that folded Reed–Solomon codes *achieve list decoding capacity*. In Figure 5.1 the speed with which (5.1) converges to $1 - R$ is illustrated.

In [26] Guruswami managed to extend the above mentioned folding operation to certain algebraic-geometry codes. This resulted in a class of folded algebraic-geometry codes, that contains folded Reed–Solomon codes as a special case. Currently these codes are the only known explicit codes that achieve list decoding capacity.

5.1.2 Folded Reed–Solomon codes

Informally a folded Reed–Solomon code simply *is* a Reed–Solomon code over some field $\mathbb{F}_q$, but viewed as a code over a larger alphabet by identifying m consecutive positions in the Reed–Solomon code with an element in $\mathbb{F}_{q^m}$.

Definition 5.1 (Folded Reed–Solomon code)**.** Let q, m, N, k be parameters such that $k \le mN \le q-1$, and let α be a primitive element of $\mathbb{F}_q$. We define the folded Reed–Solomon code with parameters q, m, N, k to be the set of $m \times N$ arrays of the form

$$\begin{bmatrix} f(1) & f(\alpha^m) & \cdots & f(\alpha^{m(N-1)}) \\ f(\alpha) & f(\alpha^{m+1}) & \cdots & f(\alpha^{m(N-1)+1}) \\ \vdots & \vdots & \ddots & \vdots \\ f(\alpha^{m-1}) & f(\alpha^{2m-1}) & \cdots & f(\alpha^{mN-1}) \end{bmatrix},$$

where $f(x) \in \mathbb{F}_q[x]$ is a polynomial of degree at most $k-1$. Using any fixed $\mathbb{F}_q$–vector space isomorphism between $(\mathbb{F}_q)^m$ and $\mathbb{F}_{q^m}$, we can consider the columns of the above array as elements of $\mathbb{F}_{q^m}$, and therefore we can consider the folded Reed–Solomon code as a code of length N over $\mathbb{F}_{q^m}$.

The identification of $(\mathbb{F}_q)^m$ and $\mathbb{F}_{q^m}$ may be viewed as *folding* a vector in $(\mathbb{F}_q)^m$ to a single element in $\mathbb{F}_{q^m}$. This is what gives folded Reed–Solomon codes their name. In the following we will call m the folding parameter of the code. We record the basic properties of folded Reed–Solomon codes in the next proposition [28, Prop. 1].

Proposition 5.2. *The folded Reed–Solomon code, with parameters as in Definition 5.1, is a (non-linear) code over $\mathbb{F}_{q^m}$ of length N, rate $R = \frac{k}{Nm}$ and minimum distance $d = N - \lceil \frac{k}{m} \rceil + 1$.*

Proof. The rate of the folded Reed–Solomon code with parameters N, k and m is

$$R = \frac{\log_{q^m}(q^k)}{N} = \frac{k}{Nm}.$$

To prove the claim on the minimum distance, assume that two codewords agree in at least $\lceil \frac{k}{m} \rceil$ positions. Then the two polynomials generating these codewords must agree in at least $m\lceil \frac{k}{m} \rceil \ge k$ points (in $\mathbb{F}_q$), which implies that they are equal. Thus the minimum distance is at least $d \ge N - \lceil \frac{k}{m} \rceil + 1$. On the other hand, the polynomial $f(x) = \prod_{i=1}^{k-1}(x - \alpha^{i-1})$ generates a codeword that agrees with the zero codeword in $\lfloor \frac{k-1}{m} \rfloor$ positions, and hence

$$d \le N - \left\lfloor \frac{k-1}{m} \right\rfloor \le N - \left(\frac{k-1-(m-1)}{m} \right) = N - \frac{k}{m} + 1.$$

Therefore
$$N - \left\lceil \frac{k}{m} \right\rceil + 1 \leq d \leq N - \frac{k}{m} + 1,$$
and since d is an integer, we conclude that $d = N - \lceil \frac{k}{m} \rceil + 1$ as claimed. □

Remark 5.3. The name *folded Reed–Solomon codes* was first used by Krachkovsky in [40] for a class of codes similar to those in Definition 5.1. Despite the similarities there are however crucial differences in the way the folding operation is done for the two types of codes. In the codes from Definition 5.1 the sub-positions in a column $(f(\alpha^{jm}), f(\alpha^{jm+1}), \ldots, f(\alpha^{jm+m-1}))$ of a codeword, are evaluations of $f(x)$ at *consecutive* powers of α. Krachkovsky folds a Reed–Solomon code "in the other direction" so that the j-th column in a codeword reads $(f(\alpha^{j}), f(\alpha^{j+N}), \ldots, f(\alpha^{j+(m-1)N}))$. Later we will see that having evaluations of $f(x)$ at consecutive powers of α in the columns of a codeword, is important for the list decodability of the codes in Definition 5.1.

Krachkovsky introduced the folded Reed–Solomon codes as a tool for correcting errors occuring in large bursts. Similarly we may view the codes in Definition 5.1 as an extension of ordinary Reed–Solomon codes capable of handling $\mathbb{F}_q$–errors occuring in bursts of length m.

5.2 Decoding folded Reed–Solomon Codes

The basic principle underlying the list decoder for folded Reed–Solomon codes from [27–29], is essentially the same as in the G–S algorithm for Reed–Solomon codes: First in an *interpolation step* a special interpolation polynomial is found, and next in a *root-finding step* a certain type of roots in this polynomial is computed. These roots will be the decoder's candidate list of transmitted words. The key difference between list decoders for Reed–Solomon codes and folded Reed–Solomon codes, is that the interpolation polynomial in the latter is allowed to have more variables. More specifically, it is allowed to be an element in $\mathbb{F}_q[x, \mathbf{T}] = \mathbb{F}_q[x, T_1, \ldots, T_v]$, where $v \leq m$ is the decoders so-called *interpolation parameter* that also appears in Equation (5.1). Below we describe the list decoder for the folded Reed–Solomon codes.

5.2.1 A list decoder for folded Reed–Solomon codes

As mentioned above the first step of the list decoder is to compute an interpolation polynomial, which is an element in $\mathbb{F}_q[x, \mathbf{T}] = \mathbb{F}_q[x, T_1, \ldots, T_v]$.

We will measure the degree of polynomials in this ring with respect to the weighted degree $\deg_{\mathbf{w}}$ from Definition 2.6, where $\mathbf{w} = (k-1, \dots, k-1) \in \mathbb{N}^v$. This means that for a monomial $x^i\mathbf{T}^{\mathbf{j}}$ in $\mathbb{F}_q[x, \mathbf{T}]$ we take its weighted degree to be

$$\deg_{\mathbf{w}} \left(x^i\mathbf{T}^{\mathbf{j}}\right) = i + (k-1)\sum_{h=1}^{v} j_h. \tag{5.2}$$

We now describe the interpolation polynomials used in the list decoder for folded Reed–Solomon codes. Similarly to the G–S algorithm the first variable x of an interpolation polynomial plays a special role, and interpolates through the powers of α (the primitive element in $\mathbb{F}_q$), while the remaining variables interpolate through the positions in the received word. In a folded Reed–Solomon code each position consists of m sub-positions from an ordinary Reed–Solomon code. Therefore we can let the variables $T_1, \dots, T_v$ interpolate through v consecutive sub-positions for each position in the folded code. This is a central idea in the list decoder for folded Reed–Solomon codes, and below we investigate its consequences. But first we make the notion of an interpolation polynomial precise. As in the G–S algorithm, the interpolation polynomial will depend on a *multiplicity parameter* s.

Definition 5.4 (Interpolation polynomial)**.** Let s be a multiplicity parameter, $v \le m$ an interpolation parameter, and let $\mathbf{y}$ be the received word

$$\mathbf{y} = \begin{bmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,N-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m-1,0} & y_{m-1,1} & \cdots & y_{m-1,N-1} \end{bmatrix}, \tag{5.3}$$

then a non-zero polynomial $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$ is said to be an (s, v)–*interpolation polynomial* for $\mathbf{y}$ if Q has a zero of multiplicity at least s in each of the points

$$\left(\alpha^{i+mj}, y_{i,j}, y_{i+1,j}, \dots, y_{i+v-1,j}\right),$$

for all $0 \le i \le m - v$ and $0 \le j < N$.

With this definition at hand we can now describe the list decoder for folded Reed–Solomon codes. Let there be given a folded Reed–Solomon code with parameters N, k, m, s and v. The list decoder then proceeds as follows:

1. **Interpolation step:** Let $\mathbf{y} \in (\mathbb{F}_{q^m})^N$ be a received word and say that the number of errors to be corrected is τ. Compute an (s, v)–interpolation polynomial $Q(x, \mathbf{T})$ for $\mathbf{y}$ of smallest possible weighted degree (with respect to $\deg_{\mathbf{w}}$).

2. **Root-finding step:** Compute and output all polynomials $f(x)$ such that $\deg(f) < k$ and
$$Q(x, f(x), f(\alpha x), \ldots, f(\alpha^{v-1}x)) = 0. \tag{5.4}$$

The background and motivation for the root-finding step, and in particular for Equation (5.4), will be given in Proposition 5.7. In the next section we analyse the performance of this list decoder. In this analysis we will see that codewords that are "close" to $\mathbf{y}$ will be on the list computed in the above two steps. On the other hand the list can also contain spurious elements that do not correspond to close-by codewords. Thus one could add a *pruning-step* after the steps described above, in which a polynomial $f(x)$ computed in the root-finding step is discarded if it does not generate a codeword "close" to $\mathbf{y}$. This last condition can be checked by simply encoding $f(x)$ and counting the agreement between the resulting codeword and $\mathbf{y}$. Such a pruning-step will however not change the performance guarantees of the list decoder, and hence in the following we neglect this step.

5.2.2 Analysis of the list decoder

In the interpolation step, we are interested in interpolation polynomials of small weighted degree. The next proposition gives a sufficient condition for the existence of an interpolation polynomial of weighted degree less than Δ.

Proposition 5.5. *Let $\mathbf{y}$ be a received word as in (5.3). Let Δ be an integer and let $\ell = \lfloor \frac{\Delta-1}{k-1} \rfloor$. If*
$$\binom{v+s}{v+1} N(m-v+1) < \Delta \binom{\ell+v}{v} - (k-1)v\binom{\ell+v}{v+1}. \tag{5.5}$$
then there exists an interpolation polynomial for $\mathbf{y}$ of weighted degree strictly less than Δ and total degree at most ℓ in the variables $\mathbf{T}$.

Proof. Using Hasse–derivatives (see Definition 2.3), the condition on a polynomial in $v+1$ variables to have a zero of multiplicity s in a single point, can be formulated as $\binom{v+s}{v+1}$ linear constraints on the coefficients of Q. Therefore the interpolation conditions in Definition 5.4, may be formulated as a linear system of $\binom{v+s}{v+1}N(m-v+1)$ equations. The number of unknowns in this system is equal to the number of monomials of weighted degree strictly less than Δ, and we now bound this quantity. There are $\binom{a+v-1}{v-1}$ ways to write the number a as a sum of v non-negative integers. Hence the number of monomials of weighted degree strictly less than Δ is

$$\sum_{i=0}^{\ell}(\Delta - i(k-1))\binom{i+v-1}{v-1}$$
$$= \Delta \sum_{i=0}^{\ell}\binom{i+v-1}{v-1} - (k-1)v\sum_{i=0}^{\ell}\binom{i+v-1}{v}$$

Using the upper-index summation binomial identity [23, p. 160], we see that this quantity is equal to the right-hand side of (5.5). Therefore under the assumptions of the proposition, the linear system has strictly more unknowns than equations, and hence it has a non-zero solution. This solution in turn gives a non-zero polynomial satisfying the interpolation constraints of weighted degree strictly less than Δ, and the proposition follows. □

Using the above proposition we can derive the following closed-form expression of a Δ for which an interpolation polynomial of weighted degree less than Δ is guaranteed to exist.

Corollary 5.6. *Let there be given parameters N, k, m, s and v of a folded Reed–Solomon code. For a word $\mathbf{y} \in (\mathbb{F}_{q^m})^N$ there exists an interpolation polynomial for $\mathbf{y}$ of weighted degree at most*

$$\Delta = \left\lfloor \sqrt[v+1]{N(m-v+1)(k-1)^v \prod_{i=0}^{v}(s+i)} \right\rfloor. \tag{5.6}$$

Proof. The corollary will follow if we can show that the Δ prescribed in (5.6) satisfies the inequality in Equation (5.5). Multiplying this inequality by $(v+1)!(k-1)^v$ we get

$$N(m-v+1)(k-1)^v \prod_{i=0}^{v}(s+i) <$$
$$((v+1)\Delta - \ell(k-1)v) \cdot \prod_{i=1}^{v}((\ell+i)(k-1)). \tag{5.7}$$

By definition of ℓ we have $\ell(k-1) < \Delta$ and $(\ell+i)(k-1) \geq \Delta$ for $i \geq 1$. Therefore the right hand side of Equation (5.7) is lower bounded by Δ^{v+1}. This implies that if we choose Δ as in Equation (5.6), then Equation (5.5) will be satisfied. □

Let $\mathbf{y}$ be a received word of a folded Reed–Solomon code. The following proposition shows that an interpolation polynomial of sufficiently low weighted degree "captures" the codewords that agree with $\mathbf{y}$ in at least $N-\tau$ positions, if τ is not too big.

Proposition 5.7. *Let $\mathbf{y} \in (\mathbb{F}_{q^m})^N$ be a received word, and assume that $\mathbf{y}$ agrees with the folded Reed–Solomon codeword generated by $f(x)$, in at least $N-\tau$ positions. Let Q be an (s,v)–interpolation polynomial for $\mathbf{y}$ of weighted degree strictly less than Δ, then if*

$$\tau \leq N - \frac{\Delta}{s(m-v+1)} \tag{5.8}$$

it holds that $Q(x, f(x), f(\alpha x), \dots, f(\alpha^{v-1}x)) = 0$.

Proof. Consider the univariate polynomial

$$\tilde{Q}(x) = Q(x, f(x), f(\alpha x), \dots, f(\alpha^{v-1}x)).$$

For each of the at least $N-\tau$ positions where $\mathbf{y}$ and the codeword generated by $f(x)$ agree, it follows from the definition of an interpolation polynomial that $\tilde{Q}(x)$ has a zero of multiplicity s in $m-v+1$ points. This means that $\tilde{Q}(x)$ has at least $(N-\tau)(m-v+1)s$ zeroes, counting multiplicities, and by the assumption on τ this quantity is at least Δ. On the other hand, since the weighted degree of Q is strictly less than Δ it follows that $\deg(\tilde{Q}(x)) < \Delta$. Hence the number of zeroes of $\tilde{Q}(x)$ is larger than its degree, and thus it must be the zero polynomial. □

Putting Corollary 5.6 and Proposition 5.7 together, we get that the list decoder for folded Reed–Solomon codes works up to error-radii satisfying

$$\frac{\tau}{N} < 1 - \left(\frac{k-1}{Nm}\right)^{\frac{v}{v+1}} \left(\frac{m}{m-v+1}\right)^{\frac{v}{v+1}} \sqrt[v+1]{\prod_{i=0}^{v}\left(1+\frac{i}{s}\right)} - \frac{1}{N}. \tag{5.9}$$

We can choose the parameters m and v such that the factor $\frac{m}{m-v+1}$ is close to one, and furthermore we can also choose s large enough that the factor

$$\sqrt[v+1]{\prod_{i=0}^{v}\left(1+\frac{i}{s}\right)},$$

is also close to one. For large values of N, the quantity $\frac{k-1}{Nm}$ is close to the rate R, and thus if we let N tend to infinity, then asymptotically the expression in Equation (5.9) essentially reduces to $1 - R^{\frac{v}{v+1}}$. Also, letting v be large this quantity tends to $1 - R$, which as mentioned in the introduction, is the best possible relative decoding radius for any list decoder. Formalizing this choice of parameters, one can prove the first item of the following theorem, see [29, Thm. 4.4] for details.

Theorem 5.8. *Let there be given $R \in [0, 1]$ and $\varepsilon > 0$. There exist parameters s, ℓ, v, m and an integer N_0 (depending on R and ε), such that: For every $N \geq N_0$ for which $N \cdot Rm$ is an integer, it holds that the folded Reed–Solomon code of rate R with multiplicity parameter s, degree bound ℓ, interpolation parameter v, folding parameter m and length N, is such that:*

- *The error-correcting radius of C is $1 - R - \varepsilon$.*
- *There exist explicit algorithms for the interpolation step and the root-finding step of the list decoder for C, both with running times polynomial in the code length.*

In Sections 5.3 and 5.4 we will prove the second item of the theorem.

Remark 5.9. Say that a sender and a receiver has agreed to protect their communication over a noisy channel with the use of folded Reed–Solomon codes. In the following we will only consider the scenario in which the sender and receiver has fixed the rate at which they wish to communicate, and furthermore that they have fixed how close to the optimal error-correcting radius $1-R$ they wish the received words to be list decoded. In the language of Theorem 5.8 this means that we will assume that R and ε are given and fixed. By Theorem 5.8 this in turn means that all the parameters s, ℓ, v and m are *constants* that are independent of the code length N. We will use this observation several times in the following.

The fixed-rate scenario described above is relevant if the rate of the communication channel does not vary much with time. Furthermore, since implementing encoders and decoders in a scenario in which the rate is allowed to vary, is often more involved than if the rate is kept constant, the fixed-rate scenario is frequently used in practice.

Remark 5.10. The code length N of a folded Reed–Solomon code satisfies $q \leq Nm + 1$, and hence the fact that the folding parameter m is constant implies that a quantity which is polynomial in q is also polynomial in the code length N.

5.3 The interpolation step

In the proof of Proposition 5.5 we saw that linear algebra can be used to compute the interpolation polynomial used in the list decoder for folded Reed–Solomon codes. Furthermore, we saw that the number of linear equations in this system is $\binom{v+s}{v+1}N(m-v+1)$, and that it has one more variables than equations. Hence the complexity of the linear algebra approach for computing interpolation polynomials, is

$$\mathcal{O}\left(s^{3(v+1)}(Nm)^3\right). \tag{5.10}$$

This shows that the complexity of the interpolation step of the list decoder is polynomial in the code length. From Remark 5.9 we have that the multiplicity parameter s is constant. Thus in principle we could drop s from the above complexity estimate. However, we have chosen to keep it since it gives information about the algorithm's dependence this parameter. In [29] it is shown that the parameters s and ℓ given by Theorem 5.8 are such that

$$s \approx \ell \sqrt[v+1]{R}. \tag{5.11}$$

This means that if the rate is not too close to zero then s and ℓ are of the same order of magnitude. To ease the comparison of the estimate in (5.10) to that of other algorithms, we will use the fact that (5.11) implies $s = \mathcal{O}(\ell)$. By the above comments, we see that the constant hidden in this approximation is not very big. Using this estimate of s, we get that the complexity of the linear algebra approach is

$$\mathcal{O}\left(\ell^{3(v+1)}(Nm)^3\right).$$

Below we will show how our general interpolation algorithm from Section 4.1 can be applied to get a different method for tackling the interpolation step. The resulting algorithm will have complexity

$$\begin{aligned}&\mathcal{O}\left(\ell^{3v}\mathrm{M}\left(\ell^{v+1}Nm\right)\log(\ell^{v+1}Nm)\right)\\ &\qquad= \mathcal{O}\left(\ell^{4v+1}Nm\log^2(\ell^{v+1}Nm)\log\log(\ell^{v+1}Nm)\right),\end{aligned} \tag{5.12}$$

where $\mathrm{M}(t) = \mathcal{O}(t\log(t)\log\log(t))$ denotes the complexity of multiplying two polynomials of degree t, see Section 3.2.3. Thus our algorithm achieves a better dependence on Nm than the linear algebra algorithm, at the cost of a worse dependence on v and ℓ (unless $v = 1$).

We now show that the interpolation problem for folded Reed–Solomon codes posed by Definition 5.4 is a special case of Problem 2.8. In Section 2.5 we saw that the projective line $\mathbb{P}^1_{\mathbb{F}_q}$ is a simple C_{ab} curve with $\delta = 0$ and $\gamma = 1$, and that the rational places of $\mathbb{F}_q(\mathbb{P}^1_{\mathbb{F}_q}) = \mathbb{F}_q(x)$ different from P_∞, are in bijective correspondence with the elements of $\mathbb{F}_q$. Below we will therefore take the vector of interpolation points $\mathbf{P}$ to be a vector with entries in $\mathbb{F}_q$. Say we have a received word $\mathbf{y}$ as in Equation (5.3), and that we wish to correct τ errors by finding an (s, v)–interpolation polynomial for $\mathbf{y}$. Consider the interpolation problem $(\mathbf{P}, \mathbf{y}', s, \ell, \mathbf{w}, \Delta)$ defined as follows:

- **Interpolation points:** The vector $\mathbf{P} \in \mathbb{F}_q^{N(m-v+1)}$ is indexed by pairs (i, j) in $\{0, \dots, m - v\} \times \{0, \dots, N - 1\}$, and the (i, j)-th coordinate is $(\mathbf{P})_{i,j} = \alpha^{i+mj}$.
- **Interpolation values:** The vector (of vectors) $\mathbf{y}'$ is also indexed by pairs (i, j) in $\{0, \dots, m-v\} \times \{0, \dots, N-1\}$, and the (i, j)-th coordinate is $(\mathbf{y}')_{i,j} = (y_{i,j}, y_{i+1,j}, \dots, y_{i+v-1,j})$.
- **Multiplicity parameter:** s.
- **Weight vector:** $\mathbf{w} = (k - 1, k - 1, \dots, k - 1) \in \mathbb{N}^v$.
- **Weighted degree bound:** $\Delta = s(N - \tau)(m - v + 1)$.
- **Degree bound:** $\ell = \lfloor \frac{\Delta - 1}{k-1} \rfloor$.

The above parameters are chosen exactly such that a polynomial $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$ is an (s, v)–interpolation polynomial for $\mathbf{y}$ if and only if it is a valid interpolation polynomial for $(\mathbf{P}, \mathbf{y}', s, \ell, \mathbf{w}, \Delta)$. Therefore we can use Algorithm 3 from Section 4.1 to compute interpolation polynomials in the list decoder for folded Reed–Solomon codes. Furthermore, by Theorem 4.2 this can be done in the following complexity:

Proposition 5.11 (Interpolation complexity)**.** *The interpolation step of the list decoding algorithm for folded Reed–Solomon codes can be done with Algorithm 3 in complexity*

$$\mathcal{O}\left(\ell^{3v} M\left(\ell^{v+1} N m\right) \log(\ell^{v+1} N m)\right).$$

5.4 The root-finding step

In this section we describe two methods for tackling the root-finding step of the list decoder for folded Reed–Solomon codes. In [27] Guruswami and

Rudra gave an elegant translation of the problem of finding roots (in $\mathbb{F}_q[x]$) satisfying Equation (5.4) to a problem of finding roots (in a finite field of size $\mathbb{F}_{q^{q-1}}$) of a univariate polynomial. Furthermore, they used this translation to derive an algorithm for the root-finding step. We describe this algorithm in Section 5.4.1.

In Section 5.4.2 we present an alternative method for solving the root finding step using Hensel–lifting.

5.4.1 Root-finding via evaluation in $\mathbb{F}_{q^{q-1}}$

In this section we show how the root-finding step of the list decoding algorithm for folded Reed–Solomon codes can done. We review the root-finding method of [27] and in particular we show that a list decoder using this method, always returns a list whose size is polynomial in the length of the code.

In the following we will call a polynomial $f(x)$ satisfying

$$Q(x, f(x), f(\alpha x), \ldots, f(\alpha^{v-1}x)) = 0, \tag{5.13}$$

a $\mathbf{T}$–root of Q. In the root-finding step we are required to extract enough information from the relation (5.13) to determine the $\mathbf{T}$–root $f(x)$. A thing that one can always do with such a relation is to evaluate it at some point $\beta \in \mathbb{F}_{q^b}$, for some $b \geq 1$. A priori we have no knowledge of the value of the vector $(f(\beta), \ldots, f(\alpha^{v-1}\beta))$ – it may be any of the q^{bv} values in $(\mathbb{F}_{q^b})^v$. But if $Q(\beta, T_1, \ldots, T_v)$ is non-zero, then it follows from Equation (5.13) that $(f(\beta), \ldots, f(\alpha^{v-1}\beta))$ is constrained to be among the $\ell q^{b(v-1)}$ possible roots in this polynomial. Thus if $\ell < q^b$ we have extracted *some* information about the $\mathbf{T}$–root $f(x)$, but not quite enough to determine it completely. However, thoughts along those lines, together with a wonderful algebraic coincidence, can actually be used to give a method for solving the root-finding step, as done in [27]. This method is based on the following observation.

Proposition 5.12. *Let α be a primitive element of $\mathbb{F}_q$. Then the polynomial $x^{q-1} - \alpha$ is irreducible in $\mathbb{F}_q[x]$.*

Proof. See [44, Thm. 3.75]. $\square$

Let $\beta \in \mathbb{F}_{q^{q-1}}$ be a root of the polynomial $x^{q-1} - \alpha$ and let $f(x)$ and $g(x)$ be polynomials in $\mathbb{F}_q[x]$ of degree at most $k-1$. Since $x^{q-1} - \alpha$ is irreducible, β can not be a root in any polynomial in $\mathbb{F}_q[x]$ of degree strictly less than

$q-1$. Since we have $k-1 < Nm \leq q-1$ this means that if $f(\beta) = g(\beta)$ then $f(x) = g(x)$. We may express this property of β by saying that evaluation at β distinguishes polynomials of degree at most $k-1$ (over $\mathbb{F}_q$). Furthermore, by definition β has another property, namely that for any $b \geq 0$,

$$f(\beta)^{q^b} = f(\beta^{q^b}) = f\left((\alpha\beta)^{q^{b-1}}\right) = f\left(\alpha\beta^{q^{b-1}}\right) = f\left(\alpha^2\beta^{q^{b-2}}\right) = \cdots = f(\alpha^b\beta).$$

It is these two properties of β that together give a method for solving the root-finding step. By evaluating (5.13) at β we get

$$0 = Q(\beta, f(\beta), f(\alpha\beta), \ldots, f(\alpha^{v-1}\beta)) = Q\big(\beta, f(\beta), f(\beta)^q, \ldots, f(\beta)^{q^{v-1}}\big),$$

and hence $f(\beta)$ is a root of the univariate polynomial $Q(\beta, T, T^q, \ldots, T^{q^{v-1}}) \in \mathbb{F}_{q^{q-1}}[T]$. Thus, in summary, we have that each root $f(x) \in \mathbb{F}_q[x]$ of (5.13) implies a root $f(\beta) \in \mathbb{F}_{q^{q-1}}$ of $Q(\beta, T, T^q, \ldots, T^{q^{v-1}})$, and that $f(\beta)$ determines $f(x)$ completely. To turn these observations into an algorithm for the root-finding step, we need to make sure that $Q(\beta, T, T^q, \ldots, T^{q^{v-1}})$ is not the zero-polynomial. We do that in two steps:

- We can write $Q(x, \mathbf{T}) = (x^{q-1} - \alpha)^b \tilde{Q}(x, \mathbf{T})$, where $\tilde{Q}(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$ and $\tilde{Q}(\beta, \mathbf{T}) \neq 0$. Any $\mathbf{T}$–root of Q is also a $\mathbf{T}$–root of $\tilde{Q}$, and thus in any further computations we may use $\tilde{Q}$ instead of Q. In other words we may without loss of generality assume that $Q(\beta, \mathbf{T})$ is non-zero.

- The univariate polynomial $Q(\beta, T, T^q, \ldots, T^{q^{v-1}})$ is zero if and only if $Q(\beta, T_1, \ldots, T_v)$ is an element of the ideal

$$\langle T_1^q - T_2, T_2^q - T_3, \ldots, T_{v-1}^q - T_v \rangle \subseteq \mathbb{F}_{q^{q-1}}[\mathbf{T}].$$

  The polynomials $\{T_1^q - T_2, T_2^q - T_3, \ldots, T_{v-1}^q - T_v\}$ form a Gröbner basis for this ideal with respect to some term order extending total degree. Hence any polynomial in the ideal must have total degree at least q. In the following we will assume that $\ell < q$. This means that the total degree of $Q(\beta, \mathbf{T})$ is strictly less than q, and hence the above shows that in this case $Q(\beta, T, T^q, \ldots, T^{q^{v-1}})$ is non-zero. Since we assume that the rate R is fixed, it follows from Remark 5.9 that the parameter ℓ is a constant independent of the code length, and thus the assumption $\ell < q$ can be *made* to holds by choosing the code length, and hence also q, large enough.

We are now ready to formalize the above observations into an algorithm for the root-finding step. We state the resulting algorithm in pseudo code in Algorithm 4.

Algorithm 4 Algorithm for root-finding via evaluation in $\mathbb{F}_{q^{q-1}}$.

```
Input: An integer $k < q$ and a polynomial $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$.
Output: The list of polynomials $f(x)$ of degree at most $k-1$ such that

        $Q(x, f(x), f(\alpha x), \dots, f(\alpha^{v-1}x)) = 0.$

 1: Write $Q(x, \mathbf{T}) = \tilde{Q}(x, \mathbf{T})(x^{q-1} - \alpha)^b$, with $b$ largest possible.
 2: Compute the set of roots $F$ of $\tilde{Q}(\beta, T, T^q, \dots, T^{q^{v-1}})$ over $\mathbb{F}_{q^{q-1}}$.
 3: Let $L \leftarrow \emptyset$.
 4: for all $f \in F$ do
 5:    Write $f$ as $f(\beta) = f_0 + f_1\beta + \dots + f_l\beta^l$ with $l < q-1$.
 6:    if $l \leq k-1$ and $\tilde{Q}(x, f(x), f(\alpha x), \dots, f(\alpha^{v-1}x)) = 0$ then
 7:       Add $f(x)$ to $L$.
 8:    end if
 9: end for
10: return $L$
```

The polynomial $Q(\beta, T, T^q, \dots, T^{q^{v-1}})$ has degree at most ℓq^{v-1}, and therefore the size of the list L computed by Algorithm 4 is at most

$$|L| \leq \ell q^{v-1}. \tag{5.14}$$

Since v is constant we get from Remark 5.10 that the above bound on the list size is polynomial in the code length N. Thus if Algorithm 4 is used for the root-finding step in the list decoder for folded Reed–Solomon codes, then the list size is polynomially bounded, which by Definition 1.1 is a necessary property of a list decoder. We also need to check that the running time of Algorithm 4 is polynomial in the code length.

Proposition 5.13. *The running time of Algorithm 4 is polynomial in q^v.*

Proof sketch. The complexity of Algorithm 4 is dominated by the complexity of line 2. In [11] Berlekamp gives a deterministic algorithm for computing the roots of a univariate polynomial over a finite field, with a complexity that is polynomial in the polynomial degree and the logarithm of the field size. The polynomial $Q(\beta, T, T^q, \dots, T^{q^{v-1}})$ has degree at most ℓq^{v-1} and hence the complexity of line 2 is polynomial in q^{v-1} and $q \log(q)$. This proves the proposition. □

The proposition proves the second item of Theorem 5.8, and thus we have now showed that folded Reed–Solomon codes can be list decoded arbitrarily close to the optimal error-radius $1 - R$, when its parameters are chosen appropriately.

5.4.2 Root-finding via Hensel–lifting

Although Proposition 5.13 promises a polynomial running time of the root-finding algorithm in Algorithm 4, the fact that the algorithm works over the exponentially large finite field $\mathbb{F}_{q^{q-1}}$, makes practical implementations of this algorithm difficult. In this section we consider an alternative method for tackling the root-finding step, using the so-called *Hensel–lifting* technique. Our method is an extension of the root-finding algorithm for Reed–Solomon codes from [54]. A method similar to ours was independently discovered by Huang and Narayanan, and appears in [35].

We now describe the Hensel–lifting algorithm. Let there be given a polynomial $Q(x, \mathbf{T})$ and an integer $b \geq 1$. In the following a polynomial $g(x)$ will be said to be a *partial* $\mathbf{T}$–root of precision b in $Q(x, \mathbf{T})$ if

$$g(x) = f(x) \bmod x^b,$$

for some $\mathbf{T}$–root $f(x)$ of $Q(x, \mathbf{T})$. Note that the degree of a partial $\mathbf{T}$–root of precision b is at most $b-1$. Hensel–lifting is a general technique for computing such partial roots up to any precision when the roots are polynomials (as in our case) or more generally when the roots are formal power series, see [59, I.2.1]. The key idea in Hensel–lifting is to recursively *lift* a partial root of precision b to a new partial root of strictly larger precision. To get the recursive process started, one needs a partial $\mathbf{T}$–root of precision $b = 1$. From Equation (5.13) it follows that if $f(x) = \sum_{i=0}^{k-1} f_i x^i$ is a $\mathbf{T}$–root of $Q(x, \mathbf{T})$, then

$$0 = Q(x, f(x), f(\alpha x), \ldots, f(\alpha^{v-1} x)) \bmod x = Q(0, f_0, f_0, \ldots, f_0). \quad (5.15)$$

Therefore a partial root f_0 of precision one must satisfy $Q(0, f_0, \ldots, f_0) = 0$. If some power of x, say x^r, divides Q then any $\mathbf{T}$–root of Q will also be a $\mathbf{T}$–root of $x^{-r}Q$. Thus we may assume that Q is not divisible by x, or in other words that $Q(0, \mathbf{T})$ is non-zero. However, if $Q(0, \mathbf{T})$ is in the ideal

$$\langle T_1 - T_2, T_2 - T_3, \ldots, T_{v-1} - T_v \rangle \subseteq \mathbb{F}_q[\mathbf{T}],$$

then $Q(0, T, \ldots, T) = 0$, and in this case we gain no information about f_0 from (5.15). If this happens, all we have is our a priori knowledge that $f_0 \in \mathbb{F}_q$, and thus we get q partial roots of precision one. On the other hand, if $Q(0, T, \dots, T) \neq 0$, we know from (5.15) that f_0 must be among the roots of this polynomial, and since its degree is at most the total degree of $Q(0, \mathbf{T})$, we get that in this case there can be at most ℓ partial roots of precision one.

The above shows how one can constrain the possible partial roots of precision one. We now move on to show how partial roots can be lifted. Assume that $f(x) = f_0 + x\tilde{f}(x)$ is a $\mathbf{T}$–root of Q and that we know f_0. Then we have that $\tilde{f}(x)$ is a $\mathbf{T}$–root of

$$Q(x, f_0 + xT_1, f_0 + \alpha x T_2, \dots, f_0 + \alpha^{v-1} x T_v). \tag{5.16}$$

If $Q(x, \mathbf{T})$ is non-zero, then the same holds for the polynomial in (5.16) as the following lemma shows.

Lemma 5.14. *Let $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$ be a non-zero polynomial and let f be an element in $\mathbb{F}_q$, then $Q(x, f + xT_1, \dots, f + \alpha^{v-1} x T_v)$ is non-zero.*

Proof. The map $\varphi_f : \mathbb{F}_q(x)[\mathbf{T}] \to \mathbb{F}_q(x)[\mathbf{T}]$ which takes $P \in \mathbb{F}_q(x)[\mathbf{T}]$ to

$$P(x, f + xT_1, \dots, f + \alpha^{v-1} x T_v),$$

is a bijection, which maps zero to zero. Since we assume that $Q(x, \mathbf{T})$ is non-zero, the lemma follows. □

From Lemma 5.14 it follows that there exists a largest power x^r of x which divides the polynomial in Equation (5.16). Let Q_{f_0} denote the polynomial obtained by dividing the polynomial in (5.16) through by x^r, that is

$$Q_{f_0}(x, \mathbf{T}) = x^{-r} Q(x, f_0 + xT_1, \dots, f_0 + \alpha^{v-1} x T_v). \tag{5.17}$$

By definition of f_0 it holds that

$$Q(x, f_0 + xT_1, f_0 + \alpha x T_2, \dots, f_0 + \alpha^{v-1} x T_v)\Big|_{x=0} = Q(0, f_0, \dots, f_0) = 0.$$

and hence we have that $r \geq 1$. As argued above, $\tilde{f}(x)$ is a $\mathbf{T}$–root of the polynomial in (5.16), and hence it follows from Equation (5.17) that $\tilde{f}(x)$ is also a $\mathbf{T}$–root of Q_{f_0}.

Let $\Phi_k(Q)$ denote the set of partial $\mathbf{T}$–roots of Q of precision k. Using this notation we can summarize the above discussion in the following recursive expression,

$$\Phi_k(Q) \subseteq \bigcup_{f_0 \in \Phi_1(Q)} \left(f_0 + x \cdot \Phi_{k-1}(Q_{f_0})\right). \tag{5.18}$$

Mimicking this expression we *define* the list of polynomials $\Lambda_k(Q)$ by

$$\begin{aligned}\Lambda_1(Q) &= \{f_0 \in \mathbb{F}_q \mid Q(0, f_0, \ldots, f_0) = 0\},\\ \Lambda_k(Q) &= \bigcup_{f_0 \in \Lambda_1(Q)} \left(f_0 + x \cdot \Lambda_{k-1}(Q_{f_0})\right).\end{aligned} \tag{5.19}$$

By definition it holds that

$$\Phi_k(Q) \subseteq \Lambda_k(Q), \tag{5.20}$$

for all $k \geq 1$. Since we know how to compute $\Lambda_1(Q)$, we can turn Equation (5.19) into a recursive algorithm for computing $\Lambda_k(Q)$. The resulting algorithm is given in pseudo code in Algorithm 5.

Algorithm 5 Algorithm $\Lambda_k(Q)$ for root-finding via Hensel–lifting.

Input: An integer k and a polynomial $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$.
Output: A list guaranteed to contain all $\mathbf{T}$–roots of Q of precision k.

```
 1: if $k \leq 0$ then
 2:    return $\{0\}$
 3: else
 4:    Let $Q \leftarrow x^{-r}Q$, with $r$ largest possible such that $x^r$ divides $Q$.
 5:    if $Q(0, T, \ldots, T) = 0$ then
 6:       $B \leftarrow \mathbb{F}_q$
 7:    else
 8:       Let $B$ be all the distinct rational roots of $Q(0, T, \ldots, T)$.
 9:    end if
10:    return $\bigcup_{f_0 \in B} (f_0 + x\Lambda_{k-1}(Q_{f_0}))$ // Compute recursively
11: end if
```

By definition the set of all $\mathbf{T}$–roots of $Q(x, \mathbf{T})$ of degree at most $k-1$ is contained in the set of partial roots $\Phi_k(Q)$, and by Equations (5.20) this set is in turn contained in $\Lambda_k(Q)$. Thus given $Q(x, \mathbf{T})$ and k as input, the output of Algorithm 5 is guaranteed to contain the set of all $\mathbf{T}$–roots of $Q(x, \mathbf{T})$ of degree at most $k-1$. Furthermore, the output list $\Lambda_k(Q)$ can be pruned to *equal* the set of $\mathbf{T}$–roots of degree at most $k-1$, by for each element $f(x)$ in $\Lambda_k(Q)$ testing whether Equation (5.13) is satisfied, and if this is not the case, remove $f(x)$ from $\Lambda_k(Q)$. Thus, altogether we conclude that Algorithm 5 provides a method for computing $\mathbf{T}$–roots, which means that it can be used to handle the root-finding step of the list decoder for folded Reed–Solomon codes. In the following we will refer to this algorithm as the *Hensel–lifting algorithm.*

5.4.3 Comparison of the root-finding methods

In this section we compare the two methods for the root-finding step, described in the two previous sections. In Section 5.4.1 we saw that Algorithm 4 returns a list of size at most ℓq^{v-1}. We begin our comparison with an example, showing that the same bound does not hold for the Hensel–lifting method in Algorithm 5.

Example 5.15. In this example we will investigate the call-tree of the recursive calls made by Algorithm 5, when it is applied to compute $\Lambda_3(Q)$ (i.e. partial $\mathbf{T}$–roots of the form $f(x) = f_0 + f_1x + f_2x^2$), where Q is the polynomial defined as follows: Let $\mathbb{F}_q = \mathbb{F}_{r^2}$ be a finite field of square order, and let α be a primitive element of $\mathbb{F}_q$. Then $Q \in \mathbb{F}_q[x, T_1, T_2]$ is the polynomial

$$Q(x,T_1,T_2) = x^{r+1}\left(\left(\frac{\alpha^2T_1 - T_2}{\alpha^2-1}\right)^r + \frac{\alpha^2T_1 - T_2}{\alpha^2-1}\right) - \left(\frac{T_2 - T_1}{\alpha - 1}\right)^{r+1}. \quad (5.21)$$

We have that $Q(0, T, T) = 0$, and hence the only constraint on the partial $\mathbf{T}$–roots of precision one we can get, is $\Lambda_1(Q) = \mathbb{F}_q$. We have

$$\begin{aligned} Q_{f_0}(x,T_1,T_2) &= x^{-(r+1)}Q(x, f_0 + xT_1, f_0 + \alpha xT_2) \\ &= f_0^r + f_0 + \left(\frac{\alpha x(\alpha T_1 - T_2)}{\alpha^2-1}\right)^r + \frac{\alpha x(\alpha T_1 - T_2)}{\alpha^2-1} - \left(\frac{\alpha T_2 - T_1}{\alpha - 1}\right)^{r+1} \end{aligned}$$

Let $A = \{f_0 \in \mathbb{F}_q \mid f_0^r + f_0 = 0\}$ and note that $|A| = r$. It turns out that the behaviour of the recursive calls made by the Hensel–lifting algorithm in line 10, i.e. when it is computing

$$\Lambda_3(Q) = \bigcup_{f_0 \in \mathbb{F}_q} (f_0 + x\Lambda_2(Q_{f_0})),$$

looks quite different depending on whether f_0 is in A or not (see Figure 5.2).

- First assume that $f_0 \in A$, then $Q_{f_0}(0, T, T) = T^{r+1}$ which implies that $f_1 = 0$. Therefore, in the next level of the call-tree we get the polynomial

  $$\begin{aligned} Q_{f_0,0}(x,T_1,T_2) &= x^{-2}Q_{f_0}(x, xT_1, \alpha xT_2) \\ &= x^{2(r-1)}\left(\frac{\alpha^2(T_1 - T_2)}{\alpha^2-1}\right)^r + \frac{\alpha^2(T_1 - T_2)}{\alpha^2-1} - x^{r-1}\frac{\alpha^2T_2 - T_1}{\alpha - 1}. \end{aligned}$$

  This means that $Q_{f_0,0}(0, T, T) = 0$, and hence the only restriction on the coefficient f_2 we can get, is that it is an element of $\mathbb{F}_q$. Therefore, the portion of the call-tree of $\Lambda_3(Q)$ that has values of f_0 lying in A, has a total of rq leaves. This is illustrated in the left half of Figure 5.2.

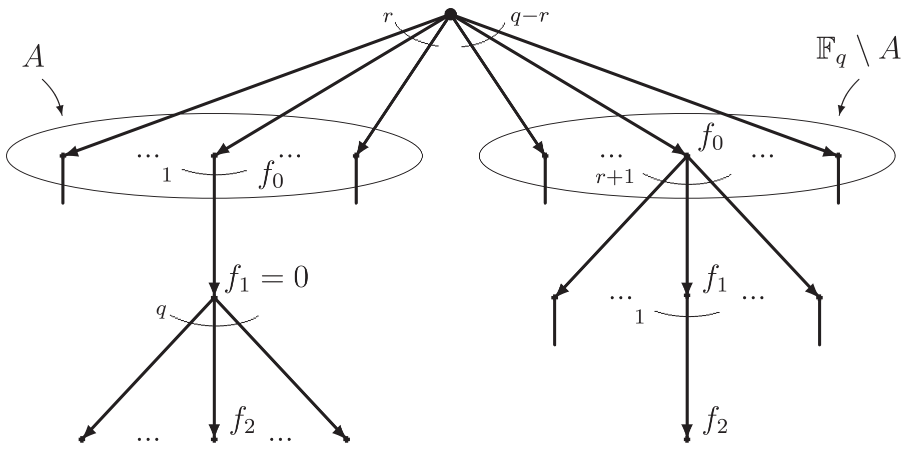


Figure 5.2: The call-tree of $\Lambda_3(Q)$, where Q is as in Equation (5.21).

- Next, assume that $f_0 \notin A$. From the expression of Q_{f_0} above, we know that $Q_{f_0}(0, T, T) = f_0^r + f_0 - T^{r+1}$. This equation resembles the defining equation for the Hermitian Curve (see Example 2.20), and we get that since $f_0^r + f_0 \in \mathbb{F}_r$ there are $r + 1$ roots of $Q_{f_0}(0, T, T)$. Let f_1 be any one of these roots, then we get that the largest b such that x^b divides $Q_{f_0}(x, f_1 + xT_1, f_1 + \alpha x T_2)$ is $b = 1$. Furthermore, by direct computation of $Q_{f_0,f_1}(x, T_1, T_2) = x^{-1} Q_{f_0}(x, f_1 + xT_1, f_1 + \alpha x T_2)$ we find that

$$Q_{f_0,f_1}(0, T_1, T_2) = f_1 \frac{\alpha(\alpha - 1)}{\alpha^2 - 1} - f_1^r \frac{\alpha^2 T_2 - T_1}{\alpha - 1}.$$

  Since $f_1 \neq 0$ we see that $Q_{f_0,f_1}(0, T, T)$ is a non-zero polynomial of degree one, and therefore it has exactly one root. This means that the portion of the call-tree of $\Lambda_3(Q)$ that has values of f_0 outside A, has a total of $(q - r)(r + 1)$ leaves. This is illustrated in the right half of Figure 5.2.

The above calculations show that at its third level, the call-tree of $\Lambda_3(Q)$ has a total of $qr + (q - r)(r + 1) = (2q - 1)r$ leaves. The total degree of Q in the variables T_1 and T_2 is $r + 1$, and hence if we use Algorithm 4 to compute **T**–roots of Q, Equation (5.14) guaranties that it returns at most $\ell q = (r + 1)q$ codewords. In general $(r + 1)q < (2q - 1)r$ and thus our example shows that the number of *partial* roots that the Hensel–lifting algorithm computes, is greater than the number of *actual* $\mathbf{T}$–roots. This fact has a number of consequences:

- The Hensel–lifting algorithm computes “spurious” partial roots that do not reflect any actual $\mathbf{T}$–roots. Furthermore, it seems difficult to bound how many such spurious roots the algorithm produces.
- This means that the running time of the Hensel–lifting algorithm can not be directly related to the number of $\mathbf{T}$–roots of the input polynomial. This in turn means that we have no “global” measure to estimate the algorithms running time, and thus the road to any such bound seems to go through “local” arguments about how the list size evolves from level to level in the call-tree.
- For general input polynomials $Q(x, \mathbf{T})$ we have unfortunately only been able to obtain local bounds that come together multiplicatively, thus yielding global bounds that are exponential in the number of levels in the call-tree, i.e. in k (and hence also in the code length). However, in Section 5.5 we shall see that if we make certain restrictions on the input polynomials, then we will be able to derive polynomial list size bounds.

We conclude the example by noting that if Algorithm 4 is used to compute $\mathbf{T}$–roots of Q, when $q = 4^2 = 16$, it returns the roots

$$\{0, 1, \alpha^5, \alpha^{10}, \alpha^4\beta^5, 1 + \alpha^4\beta^5, \alpha^5 + \alpha^4\beta^5, \alpha^{10} + \alpha^4\beta^5\} \subseteq \mathbb{F}_{q^{q-1}} = \mathbb{F}_{2^{60}}.$$

By inserting into $Q(x, T_1, T_2)$ we see that out of these roots only four correspond to actual $\mathbf{T}$–roots, namely

$$f(x) = 0, \quad f(x) = 1, \quad f(x) = \alpha^5, \quad f(x) = \alpha^{10}.$$

Example 5.16 (Running times)**.** In this example we compare the performance of the two root-finding algorithms, when they are used in the list decoder for folded Reed–Solomon codes. The comparison is made by simulating data transmission over a noisy channel employing various folded Reed–Solomon codes, that are list decoded at the receiver end.

In the simulations we have used the following folded Reed–Solomon codes: for $i \in \{2, \dots, 5\}$ we let C_i be the folded Reed–Solomon code of rate (approximately) $R = \frac{1}{5}$, with parameters $q_i = 4^i$, $m_i = 3$, $N_i = \frac{q_i - 1}{m_i}$, $v_i = 2$ and $k_i = \lceil R N_i m_i \rceil$. The parameters of the four codes are given explicitly in Table 5.1.

C_i	q_i	k_i	N_i	Δ_i	τ_i
C_2	16	3	5	12	3
C_3	64	13	21	61	10
C_4	256	51	85	247	43
C_5	1024	205	341	999	174

Table 5.1: Parameters of the folded Reed–Solomon codes $C_2, \ldots, C_5$.

We use multiplicity parameter $s = 3$ in the list decoder. For each code C_i in Table 5.1 we have used Proposition 5.5 with $s = 3$ to compute a weighted degree bound Δ_i, such that an interpolation polynomial of weighted degree strictly less than Δ_i is guaranteed to exist for any received word in $(\mathbb{F}_{q^m})^N$. These weighted degree bounds are given in Table 5.1. Furthermore, for each code we have computed the largest integer τ_i which is at most $N_i - \frac{\Delta_i}{s(m-v+1)}$. As we saw in Proposition 5.7, the list decoder for C_i is guaranteed to reconstruct a transmitted codeword when the corresponding received word is corrupted by at most τ_i errors.

For each of the codes C_i in Table 5.1, we simulate the transmission of a codeword from C_i over a noisy channel as follows:

1. Choose a polynomial in $\mathbb{F}_q[x]$ of degree strictly less than k_i at random, and encode it to a codeword $\mathbf{c} \in C_i$.
2. Generate a received word $\mathbf{y}$ by adding τ_i errors at random to $\mathbf{c}$.
3. Compute an interpolation polynomial Q for $\mathbf{y}$ of weighted degree strictly less than Δ_i.
4. Use both root-finding algorithms to compute the $\mathbf{T}$–roots of Q.

We have implemented each of the four steps above, in the computer algebra system `Magma` [13]. Since we are not concerned with the running time of the interpolation step, we have implemented step 3 using linear algebra, as described in the proof of Proposition 5.5. The source code for the implementations can be obtained from [14].

We have repeated the above four-step simulation 10 times for each code C_i, and recorded the average and standard deviation of the running times of each of the two algorithms used in the root-finding step. The results are listed in Table 5.2. Executions running longer than six hours have been terminated and this event is marked with a '$-$'. All simulations were carried out on the same computer, and thus the timings in Table 5.2 should provide a fair basis for comparing the performance of the two algorithms.

C_i	Algorithm 4		Algorithm 5 (Hensel–lifting)	
	μ	σ	μ	σ
C_2	0.070	0.00	0.01	0.02
C_3	101.71	0.73	0.22	0.27
C_4	–	–	87.42	98.49
C_5	–	–	1962.30	5986.44

Table 5.2: Running times (in seconds) of the two algorithms for the root-finding step in the list decoder for $C_2, \ldots, C_5$. The numbers are averages (μ) and standard deviations (σ) of 10 decodings.

In the simulations the transmission errors are generated randomly. From Table 5.2 we see that in this situation the performance of the Hensel–lifting algorithm seems to be *much* better than Algorithm 4. Thus despite the lack of theoretical bounds on the running time of the Hensel–lifting algorithm, it performs well in practice – at least for random errors.

We remark that the standard deviations of the running times of the Hensel–lifting algorithm reported in Table 5.2 are relatively large, i.e. of the same order of magnitude as the averages. The reason for this is the occurrence of unconstrained coefficients, which significantly increases the running time of the algorithm. For instance, all but one of the Hensel–lifting computations for the code C_5 took about a minute, while a single computation, that involved an unconstrained coefficient, took more than 5 hours.

Thus the practical performance of the Hensel–lifting algorithm is very sensitive to the occurrence of unconstrained coefficients. This is also illustrated by the running time of the Hensel–lifting algorithm when it is used to compute $\mathbf{T}$–roots of degree less than $q-1$ in the polynomial from Equation (5.21) with $q = 256$. In this computation $\sqrt{q}+1 = 17$ unconstrained coefficients are encountered, and the algorithm takes about 9 hours to finish (on the same computer on which the timings in Table 5.2 were made). On the other hand, if Algorithm 4 is used for the same task, termination does not occur within 15 hours. Thus even for a polynomial which is carefully chosen to be difficult for the Hensel–lifting algorithm, it is still faster in practice than Algorithm 4.

5.5 Bounds on the list size

In this section we will present results towards a bound on the size of the list computed by the Hensel–lifting algorithm (Algorithm 5). We will only be able to derive bounds under certain assumptions, either on the parameter v or on the multiplicities of the roots of the polynomial $Q(x, \mathbf{T})$ to which the Hensel–lifting algorithm is applied. However, the techniques used in the proofs of these results reveals much about the root-finding problem's nature, and furthermore they provide a good starting point for investigations of more general list size bounds.

5.5.1 A bound on the list size for $v \leq 2$

In this section we will derive a bound on the size of the list computed by the Hensel–lifting algorithm (Algorithm 5) when $v \leq 2$. We will derive this bound by tracking the recursive calls made by the algorithm, and to this end we need the notion of a *recursion tree* which we introduce below. By a *branch* between two nodes A and B in a tree, we will understand a sequence of nodes

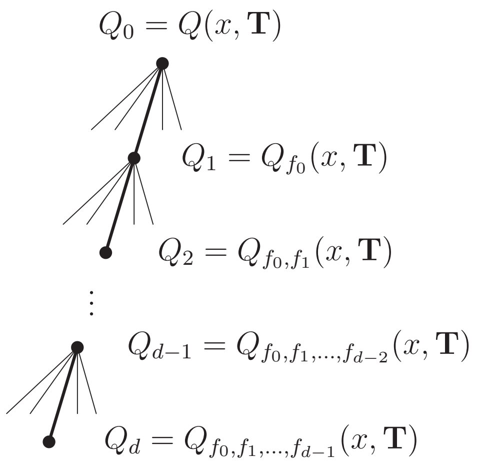
$$Q_0, Q_1, \ldots, Q_d, \tag{5.22}$$

such that $A = Q_0$, $B = Q_d$ and such that Q_i is a child of Q_{i-1} for $1 \leq i \leq d$. The *length* of the branch in Equation (5.22) is d. If there exists a branch between two nodes A and B, then we define the *distance* between A and B to be the length of this branch.

Definition 5.17 (Recursion tree). Let $Q \in \mathbb{F}_q[x, \mathbf{T}]$ be a polynomial such that $Q(0, \mathbf{T}) \neq 0$. The *recursion tree* of $\Lambda_k(Q)$ is the tree defined as follows:

- The nodes are labeled by polynomials in $\mathbb{F}_q[x, \mathbf{T}]$. The root node is Q.
- A node P is assigned child nodes according to the following rules:
  1. If $P(0, T, \ldots, T) = 0$ and if the distance from Q to P is at most $k-1$, then the child nodes of P are $P_{\alpha_1}, \ldots, P_{\alpha_q}$, where $\alpha_1, \ldots, \alpha_q$ are the q distinct elements of $\mathbb{F}_q$.
  2. If $P(0, T, \ldots, T) \neq 0$ and if the distance from Q to P is at most $k-1$, then the child nodes of P are $P_{\alpha_1}, \ldots, P_{\alpha_h}$, where $\alpha_1, \ldots, \alpha_h$ are the distinct rational roots of $P(0, T, \ldots, T)$.

  We will say that a node satisfying the requirements in item 1 is *unconstrained*.



$Q_0 = Q(x,\mathbf{T})$
$Q_1 = Q_{f_0}(x,\mathbf{T})$
$Q_2 = Q_{f_0,f_1}(x,\mathbf{T})$
$Q_{d-1} = Q_{f_0,f_1,\ldots,f_{d-2}}(x,\mathbf{T})$
$Q_d = Q_{f_0,f_1,\ldots,f_{d-1}}(x,\mathbf{T})$

Figure 5.3: Branch in the recursion tree of $\Lambda_k(Q)$

The recursion tree of $\Lambda_k(Q)$ is defined exactly such that the child nodes of Q are the polynomials given as input to Λ_{k-1} in the recursive calls made in line 10 of Algorithm 5. In Figure 5.3 a branch in this recursion tree is illustrated.

In this section we will derive a bound on the number of leaves in the recursion tree of $\Lambda_k(Q)$ when $v \leq 2$, and we will use this to obtain a bound on the size of the list returned by Algorithm 5 and on the algorithm's running time.

In the following we will say that $\mathbf{a} = (a_0, a_1, \ldots, a_v) \in \mathbb{F}_q^{v+1}$ is a zero of multiplicity r in $Q(x,\mathbf{T}) \in \mathbb{F}_q[x,\mathbf{T}]$, if r is the smallest integer such that the $(i,\mathbf{j})$-th Hasse–derivative of Q vanish in $\mathbf{a}$, i.e. $\mathcal{H}_{x,\mathbf{T}}^{(i,\mathbf{j})}(Q)(\mathbf{a}) = 0$, for all $(i,\mathbf{j})$ with $i + \sum_{h=1}^{v} j_h < r$. The following lemma shows that if we know the multiplicity of $(0, f_0, \ldots, f_0)$ as a zero in $Q(x,\mathbf{T})$, then we can determine $Q_{f_0}(0,\mathbf{T})$ explicitly, a fact that we will use repeatedly below.

Lemma 5.18. *Let $(0,\mathbf{f}_0) = (0, f_0, \ldots, f_0) \in \mathbb{F}_q^{v+1}$ be a zero in $Q(x,\mathbf{T})$ of multiplicity r. Then it holds that*

$$Q_{f_0}(0,\mathbf{T}) = \sum_{(i,\mathbf{j})\in\mathbb{N}_0^{v+1} \,:\, i+\sum_h j_h = r} \mathcal{H}_{x,\mathbf{T}}^{(i,\mathbf{j})}(Q)(0,\mathbf{f}_0)\alpha^{\sum_{h=1}^{v}(h-1)j_h}\mathbf{T}^{\mathbf{j}}. \tag{5.23}$$

Furthermore, for all index vectors $\mathbf{j}$ with $\sum_{h=1}^{v} j_h = r$, it holds that

$$\mathcal{H}_{\mathbf{T}}^{(\mathbf{j})}(Q_{f_0})(0,\mathbf{T}) = \alpha^{\sum_{h=1}^{v}(h-1)j_h}\mathcal{H}_{\mathbf{T}}^{(\mathbf{j})}(Q)(0,\mathbf{f}_0). \tag{5.24}$$

Proof. By the the Taylor–expansion formula for Hasse–derivatives we can write

$$Q(x, \mathbf{T}) = \sum_{i,\mathbf{j}} \mathcal{H}^{(i,\mathbf{j})}_{x,\mathbf{T}}(Q)(0, \mathbf{f}_0)x^i(\mathbf{T} - \mathbf{f}_0)^{\mathbf{j}}. \tag{5.25}$$

From this expression we get that

$$\begin{aligned} &Q(x, f_0 + xT_1, \dots, f_0 + \alpha^{v-1}xT_v) \\ &\qquad = \sum_{i,\mathbf{j}} \mathcal{H}^{(i,\mathbf{j})}_{x,\mathbf{T}}(Q)(0, \mathbf{f}_0)x^{i+\sum_h j_h}\alpha^{\sum_h j_h(h-1)}\mathbf{T}^{\mathbf{j}}. \end{aligned} \tag{5.26}$$

By assumption r is the smallest integer such that $\mathcal{H}^{(i,\mathbf{j})}_{x,\mathbf{T}}(Q)(0, \mathbf{f}_0) = 0$ for all $(i, \mathbf{j}) \in \mathbb{N}_0^{v+1}$ with $i + \sum_{h=1}^{v} j_h < r$. Therefore the largest power of x that divides the polynomial in (5.26), is x^r. Hence

$$Q_{f_0}(x, \mathbf{T}) = \sum_{i,\mathbf{j}} \mathcal{H}^{(i,\mathbf{j})}_{x,\mathbf{T}}(Q)(0, \mathbf{f}_0)x^{i+\sum_h j_h - r}\alpha^{\sum_h j_h(h-1)}\mathbf{T}^{\mathbf{j}}, \tag{5.27}$$

and from this equation, the expression for $Q_{f_0}(0, \mathbf{T})$ in Equation (5.23) follows. To prove the second claim of the lemma, let $\mathbf{j}$ be such that $\sum_{h=1}^{v} j_h = r$ and note that since $Q_{f_0}(x, \mathbf{T}) = Q_{f_0}(0, \mathbf{T}) + \mathcal{O}(x)$ it follows from (5.23) that

$$\mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}(Q_{f_0})(x, \mathbf{T}) = \alpha^{\sum_h (h-1)j_h}\mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}(Q)(0, \mathbf{f}_0) + \mathcal{O}(x).$$

Evaluating this expression in $x = 0$, Equation (5.24) follows. □

Corollary 5.19. *Let* $(0, \mathbf{f}_0) = (0, f_0, \dots, f_0) \in \mathbb{F}_q^{v+1}$ *be a zero in* $Q(x, \mathbf{T})$. *Then it holds that*

$$\deg_{tot}(Q_{f_0}(0, \mathbf{T})) \leq \deg_{tot}(Q(0, \mathbf{T})). \tag{5.28}$$

Furthermore, if equality holds in (5.28) then the multiplicity of $(0, \mathbf{f}_0)$ *in* $Q(x, \mathbf{T})$ *is equal to* $\deg_{tot}(Q(0, \mathbf{T}))$.

Proof. By Lemma 5.18 we have that $\deg_{\mathrm{tot}}(Q_{f_0}(0, \mathbf{T}))$ is at most the multiplicity of the zero $(0, \mathbf{f}_0)$ in $Q(x, \mathbf{T})$. Therefore we have that

$$\begin{aligned} \deg_{\mathrm{tot}}(Q_{f_0}(0, \mathbf{T})) &\leq \text{Multiplicity of } (0, \mathbf{f}_0) \text{ in } Q(x, \mathbf{T}) \\ &\leq \text{Multiplicity of } \mathbf{f}_0 \text{ in } Q(0, \mathbf{T}) \\ &\leq \deg_{\mathrm{tot}}(Q(0, \mathbf{T})). \end{aligned}$$

From this the claims of the corollary follows. □

To get a non-trivial bound on the number of leaves of a recursion tree, we need a bound on how many unconstrained nodes there can be in such a tree. The following lemma provides a step towards this.

Lemma 5.20. *Let $v \leq 2$ and let $r \geq 1$. Furthermore, if $v = 2$ assume that $r < q-1$. Let $Q_0, \dots, Q_d$ be a branch in a recursion tree, i.e. a sequence of polynomials such that $Q_i = (Q_{i-1})_{f_{i-1}}$, where $f_{i-1} \in \mathbb{F}_q$ is some zero of $Q_{i-1}(0, T, \dots, T)$. If $d < q-1$ and if*

$$\deg_{tot}(Q_i(0, \mathbf{T})) = r,$$

for $0 \leq i \leq d$, then there can be at most $\binom{r+v-1}{v-1} - 1$ distinct indices i for which Q_i is unconstrained.

Proof. First, assume that $v = 1$. By definition, there can not be any unconstrained nodes in this case. We have that $\binom{r+1-1}{1-1} - 1 = 0$, and hence the lemma holds in this special case.

Next, assume that $v = 2$. The branch $Q_0, \dots, Q_d$ can not have more than $d+1$ unconstrained nodes, and hence in the following we may assume that

$$d + 1 > \binom{r+2-1}{2-1} - 1 = r \geq 1,$$

and hence also that $d > 0$. In the remainder of the proof we will keep v as a variable (instead of specializing it to $v = 2$), since this will allow us to see exactly where the proof breaks down when $v \geq 3$. See Remark 5.21.

By assumption we have that $\deg_{\mathrm{tot}}(Q_i(0, \mathbf{T})) = \deg_{\mathrm{tot}}(Q_{i-1}(0, \mathbf{T})) = r$ for $i \geq 1$. Thus from Corollary 5.19 it follows that for $i \geq 1$, $(0, \mathbf{f}_{i-1})$ is a root of multiplicity exactly r in $Q_{i-1}(x, \mathbf{T})$. Therefore Lemma 5.18 implies that

$$Q_i(0, \mathbf{T}) = \sum_{(a, \mathbf{j})\,:\, a + \sum_h j_h = r} \mathcal{H}^{(a, \mathbf{j})}_{x, \mathbf{T}}\left(Q_{i-1}\right)(0, \mathbf{f}_{i-1}) \alpha^{\sum_{h=1}^{v}(h-1) j_h} \mathbf{T}^{\mathbf{j}}. \tag{5.29}$$

and that

$$\mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}\left(Q_i\right)(0, \mathbf{T}) = \alpha^{\sum_{h=1}^{v}(h-1) j_h} \mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}\left(Q_{i-1}\right)(0, \mathbf{f}_{i-1}), \tag{5.30}$$

for index vectors $\mathbf{j}$ with $\sum_{h=1}^{v} j_h = r$. Using Equation (5.30) recursively it follows that

$$\begin{aligned} \mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}\left(Q_i\right)(0, \mathbf{f}_i) &= \alpha^{\sum_{h=1}^{v}(h-1) j_h} \mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}\left(Q_{i-1}\right)(0, \mathbf{f}_{i-1}) \\ &= \cdots = \alpha^{i \sum_{h=1}^{v}(h-1) j_h} \mathcal{H}^{(\mathbf{j})}_{\mathbf{T}}\left(Q_0\right)(0, \mathbf{f}_0). \end{aligned} \tag{5.31}$$

Now assume, for the sake of contradiction, that there exist $r' = \binom{r+v-1}{v-1}$ indices $i_1 < i_2 < \cdots < i_{r'}$ such that

$$Q_{i_u}(0, T, \ldots, T) = 0, \tag{5.32}$$

for $1 \leq u \leq r'$. A necessary condition for the polynomial in (5.32) to vanish, is that all the coefficients to the monomials in $Q_{i_u}(0, \mathbf{T})$ of the form $\mathbf{T}^{\mathbf{j}}$ with $\sum_{h=1}^{v} j_h = r$, must sum to zero. By Equations (5.29) and (5.31) the coefficient to such a monomial $\mathbf{T}^{\mathbf{j}}$ is

$$\alpha^{i_u \sum_{h=1}^{v}(h-1)j_h} \mathcal{H}_{\mathbf{T}}^{(\mathbf{j})}(Q_0)(0, \mathbf{f}_0).$$

Consider the $r' \times r'$ matrix $\mathbf{A}$ defined by

$$[\mathbf{A}]_{u,\mathbf{j}} = \alpha^{i_u \sum_{h=1}^{v}(h-1)j_h},$$

for $1 \leq u \leq r'$ and $\mathbf{j}$ in $\{\mathbf{j} \in \mathbb{N}_0^v \mid \sum_{h=1}^{v} j_h = r\}$. The above argument shows that the vector

$$\left(\mathcal{H}_{\mathbf{T}}^{(\mathbf{j})}(Q_0)(0, \mathbf{f}_0)\right)_{\mathbf{j}} \in \mathbb{F}_q^{r'}, \tag{5.33}$$

again indexed by $\mathbf{j}$ in $\{\mathbf{j} \in \mathbb{N}_0^v \mid \sum_{h=1}^{v} j_h = r\}$, must be an element of the right kernel of $\mathbf{A}$. Let $\mathbf{j}$ and $\mathbf{j}'$ be two vectors with $\sum_{h=1}^{v} j_h = \sum_{h=1}^{v} j'_h = r$. If

$$\alpha^{\sum_{h=1}^{v}(h-1)j_h} = \alpha^{\sum_{h=1}^{v}(h-1)j'_h}$$

then, since $v = 2$ and since the order of α is $q-1$, it must hold that $j_2 \equiv j'_2 \pmod{q-1}$. By assumption we have $j_2 \leq r < q-1$ and hence this equivalence implies that $j_2 = j'_2$, which in turn means that $\mathbf{j} = \mathbf{j}'$. Therefore all the r' elements $\alpha^{\sum_{h=1}^{v}(h-1)j_h}$ with $\sum_{h=1}^{v} j_h = r$, are distinct. Since we also have

$$i_1 < i_2 < \ldots < i_{r'} \leq d < q-1,$$

it follows that $\mathbf{A}$ is a non-singular Vandermonde matrix. This in turn implies that the vector in Equation (5.33) must be zero. By Equation (5.29) this means that $Q_1(0, \mathbf{T})$ has total degree strictly less than r, which contradicts our assumptions (since $d > 0$). Thus we have proved that there can be at most $r' - 1$ indices i for which the polynomial $Q_i(0, T, \ldots, T)$ vanishes. □

Remark 5.21. If $v \geq 3$ the proof of the above lemma breaks down, because it no longer holds that the $r' = \binom{r+v-1}{v-1}$ elements $\alpha^{\sum_{h=1}^{v}(h-1)j_h}$ with $\sum_{h=1}^{v} j_h = r$, are distinct. As an example of this phenomenon let $v = 3$ and take $Q(x, \mathbf{T}) = T_1T_3 - T_2^2$. In this polynomial, there are two distinct power vectors $\mathbf{j} = (1, 0, 1)$ and $\mathbf{j}' = (0, 2, 0)$, and these satisfy

$$\alpha^{\sum_{h=1}^{3}(h-1)j_h} = \alpha^{\sum_{h=1}^{3}(h-1)j'_h} = \alpha^2.$$

In fact, for $f_0 = 0$ it holds that $Q_{f_0}(x, \mathbf{T}) = \alpha^2 Q(x, \mathbf{T})$ and therefore in the recursion tree for $\Lambda_k(Q)$ there is a branch $Q_0, \dots, Q_k$ in which *every* polynomial Q_i satisfies $\deg_{\text{tot}}(Q_i(0, \mathbf{T})) = 2$ and $Q_i(0, T, T, T) = 0$. Thus a direct extension of Lemma 5.20 to the case $v \geq 3$, is not possible.

To derive the desired bound on the size of the list returned by the Hensel–lifting algorithm we will need the following lemma, which bounds the effect of unconstrained nodes on the number of leaves in a recursion tree.

Lemma 5.22. *Let $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$ be a polynomial such that $Q(0, \mathbf{T}) \neq 0$, and let Γ be the recursion tree of $\Lambda_k(Q)$. Assume that for any branch $Q_0, \dots, Q_d$ in Γ, there are at most u indices i for which Q_i is unconstrained. Then the number of leaves in Γ is at most $\deg_{tot}(Q(0, \mathbf{T}))q^u$.*

Proof. We prove the lemma by induction on u, and begin with the induction base $u = 0$. In this case we have that $Q(0, T, \dots, T) \neq 0$. If f is a root of $Q(0, T, \dots, T)$, then by Lemma 5.18 it holds that the total degree of $Q_f(0, \mathbf{T})$ is less than or equal to the multiplicity of $(0, f, \dots, f)$ in $Q(x, \mathbf{T})$. This multiplicity is less than or equal to the multiplicity of f in $Q(0, T, \dots, T)$, and hence we have that

$$\sum_f \deg_{\text{tot}}(Q_f(0, \mathbf{T})) \leq \deg(Q(0, T, \dots, T)) \leq \deg_{\text{tot}}(Q(0, \mathbf{T})), \qquad (5.34)$$

where the sum is over all the distinct zeroes f of $Q(0, T, \dots, T)$. Let Γ_f denote the subtree of Γ rooted in the child node Q_f of Q, see Figure 5.4. There are no unconstrained nodes in Γ_f, and hence we may apply our argumentation recursively to this tree. By this recursion (or alternatively induction of the depth of Γ) we have that

$$\#\text{ leaves of }\Gamma = \sum_f (\#\text{ leaves of }\Gamma_f) \leq \sum_f \deg_{\text{tot}}(Q_f(0, \mathbf{T})),$$

where again the sums are over all the distinct zeroes f of $Q(0, T, \dots, T)$. Using this inequality together with Equation (5.34), it follows that Γ has at most $\deg_{\text{tot}}(Q(0, \mathbf{T}))$ leaves. This proves the induction base.

For the induction step assume that $u \geq 1$ and that the lemma has been proved for smaller u. Let Γ' be the subtree of Γ constructed by the following recipe:

1. Make Q the root node of Γ'.

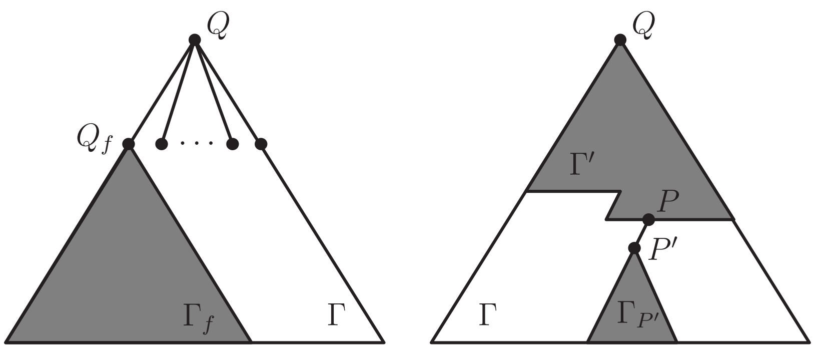


Figure 5.4: The subtrees Γ_f (left) and $\Gamma_{P'}$ (right) of the recursion tree Γ.

2. For each leaf-node P of Γ' for which $P(0, T, \dots, T) \neq 0$, add the child nodes of P in Γ (and the edges connecting them) to Γ'.
3. Repeat step 2 until no new nodes are added to Γ'.

By definition, Γ' can only have unconstrained nodes in its leaves. The polynomials in the leaf nodes of a recursion tree do not affect its size. Therefore we may apply exactly the same argument as in the induction base to the tree Γ', to get that

$$\sum_{P \text{ a leaf of } \Gamma'} \deg_{\text{tot}}(P(0, \mathbf{T})) \leq \deg_{\text{tot}}(Q(0, \mathbf{T})). \tag{5.35}$$

Let P be a leaf node of Γ'. To prove the induction step we count how many leaves the subtree of Γ rooted in P can have. First we note that if $P(0, T, \dots, T) \neq 0$, then by definition of Γ', P is also a leaf of Γ, and hence it has no successors. Next, assume that $P(0, T, \dots, T) = 0$. Then, unless the distance from P to the root is k, P will have q child nodes. Let P' denote such a child node, and let $\Gamma_{P'}$ denote the subtree of Γ rooted in P', see Figure 5.4. Any branch in $\Gamma_{P'}$ can have at most $u - 1$ unconstrained nodes, and hence it follows from the induction hypothesis that the number of leaves in $\Gamma_{P'}$ is at most $\deg_{\text{tot}}(P'(0, \mathbf{T}))q^{u-1}$. Therefore the number of leaves in the subtree of Γ rooted in P is at most

$$q \cdot \deg_{\text{tot}}(P'(0, \mathbf{T}))q^{u-1} \leq \deg_{\text{tot}}(P(0, \mathbf{T}))q^u,$$

where the inequality follows from Corollary 5.19. This means that the number of leaves of Γ is at most

$$\sum_{P \text{ a leaf of } \Gamma'} (\#\text{ leaves of subtree rooted in } P) \leq \sum_{P \text{ a leaf of } \Gamma'} \deg_{\text{tot}}(P(0, \mathbf{T}))q^u.$$

By Equation (5.35) the right hand side of the above inequality is at most $\deg_{\text{tot}}(Q(0,\mathbf{T}))q^u$, and thus the induction step follows. □

Putting the above two lemmas together we can now prove the desired bound on the size of the list returned by $\Lambda_k(Q)$.

Theorem 5.23. *Let $v \leq 2$ and let $Q \in \mathbb{F}_q[x, \mathbf{T}]$ be a polynomial for which $Q(0,\mathbf{T}) \neq 0$. Furthermore, let ℓ denote the total degree of $Q(0,\mathbf{T})$, and assume that $\ell < q-1$ if $v = 2$. If $k < q-1$ then it holds that*

$$|\Lambda_k(Q)| \leq \ell q^{\binom{\ell+v}{v}-\ell-1} = \begin{cases} \ell & \text{if } v = 1, \\ \ell q^{\binom{\ell+1}{2}} & \text{if } v = 2. \end{cases}$$

Proof. Let Γ denote the recursion tree of $\Lambda_k(Q)$. Consider a branch

$$Q = Q_0, \ldots, Q_d, \tag{5.36}$$

in Γ from the root node Q to a leaf node Q_d. Let $f_{i-1} \in \mathbb{F}_q$ be such that $Q_i = (Q_{i-1})_{f_{i-1}}$, then it follows from Corollary 5.19 that

$$\deg_{\text{tot}}(Q_{i-1}(0,\mathbf{T})) \geq \deg_{\text{tot}}(Q_i(0,\mathbf{T})),$$

for $i \geq 1$. This implies that

$$\deg_{\text{tot}}(Q_i(0,\mathbf{T})) \geq \deg_{\text{tot}}(Q_{i'}(0,\mathbf{T})), \tag{5.37}$$

for $i' \geq i$. From this it follows that the branch in Equation (5.36) can be split into subbranchs

$$Q_{\ell,0}, \ldots, Q_{\ell,d_\ell}; \ldots; Q_{i,0}, \ldots, Q_{i,d_i}; \ldots; Q_{1,0}, \ldots, Q_{1,d_1},$$

such that for $1 \leq i \leq \ell$ and $0 \leq j \leq d_i$ it holds that the total degree of $Q_{i,j}(0,\mathbf{T})$ is equal to i. From Lemma 5.20 we have that in the subbranch $Q_{i,0}, \ldots, Q_{i,d_i}$ there can be at most $\binom{i+v-1}{v-1} - 1$ nodes $Q_{i,j}$ for which $Q_{i,j}(0, T, \ldots, T) = 0$. Therefore in the branch in Equation (5.36) there can be at most

$$\sum_{i=1}^{\ell} \left(\binom{i+v-1}{v-1} - 1 \right) = \binom{\ell+v}{v} - \ell - 1,$$

unconstrained nodes. Since the branch in Equation (5.36) is arbitrary, it therefore follows from Lemma 5.22 that the number of leaves in Γ is at most

$$\deg_{\text{tot}}(Q(0,\mathbf{T}))q^{\binom{\ell+v}{v}-\ell-1} = \ell q^{\binom{\ell+v}{v}-\ell-1},$$

and the theorem follows. □

From Remark 5.9 we know that the parameters v and ℓ in the list decoder for folded Reed–Solomon codes are *constants* independent of the code length. This means that if $v = 2$, then for a given rate it is possible to choose the code length, and hence also q, large enough that the assumption $\ell < q - 1$ in Theorem 5.23 holds. If this is done, the theorem bounds the size of the list returned by the Hensel–lifting algorithm by $\ell q^{\binom{\ell+1}{2}}$, and since ℓ is constant as a function of the code length, this quantity is polynomial in q. From Remark 5.10 it therefore follows that the list size bound $\ell q^{\binom{\ell+1}{2}}$ is also polynomial in the code length.

Remark 5.24. A natural question is whether the (rather large) list size bound in Theorem 5.23 for $v = 2$ is tight, i.e. if there exist an input polynomial $Q(x, T_1, T_2)$ which results in an output list of size $\ell q^{\binom{\ell+1}{2}}$. In Section 5.4.1 we saw that when $v = 2$ the size of the list returned by Algorithm 4 is at most ℓq, and therefore one might conjecture that the same list size bound should hold for the Hensel–lifting algorithm. This is however not the case, as we saw in Example 5.15.

A crucial property of a list decoder is that its running time is polynomial in the code length (see Definition 1.1). In the following proposition we show that if $v \leq 2$ and the Hensel–lifting algorithm is used in the root-finding step, then the list decoder for folded Reed–Solomon codes has this property. Thus for $v \leq 2$ this result provides an alternative to Section 5.4.1 for proving the second item of Theorem 5.8. We will only be interested in the qualitative result, that the running time is polynomial, and not the precise polynomial bound. Therefore, we will only provide a rather rough complexity estimate.

Proposition 5.25. *Let parameters be as in Theorem 5.23. The running time of $\Lambda_k(Q)$ (Algorithm 5) is polynomial in $q^{\binom{\ell+v}{v}}$.*

Proof. Let Γ denote the recursion tree of $\Lambda_k(Q)$. The number of nodes at distance i from the root of Γ is equal to the number of leaves in the recursion tree of $\Lambda_i(Q)$. Therefore it follows from Theorem 5.23 that for $i \leq k$, the number of nodes at distance i from the root node is at most $\ell q^{\binom{\ell+v}{v}-\ell-1}$, and hence the total number of nodes in Γ is at most

$$k \cdot \ell q^{\binom{\ell+v}{v}-\ell-1} \leq \ell q^{\binom{\ell+v}{v}-\ell}.$$

This quantity is polynomial in $q^{\binom{\ell+v}{v}}$ and hence the proposition will follow if we can show that the complexity of the computations made in each node of Γ, i.e. in the body of Algorithm 5, is also polynomial in $q^{\binom{\ell+v}{v}}$. Let Q_{i-1} be a node in Γ and let $Q_i = (Q_{i-1})_{f_{i-1}}$ be one of its child nodes. By Corollary 5.19 it holds that $\deg_{\text{tot}}(Q_i(0,\mathbf{T})) \leq \deg_{\text{tot}}(Q_{i-1}(0,\mathbf{T}))$ and furthermore from the definition of $Q_i = (Q_{i-1})_{f_{i-1}}$ (see Equation (5.17)) it follows that

$$\deg_x(Q_i(x,\mathbf{T})) \leq \deg_x(Q_{i-1}(x,\mathbf{T})) + \ell.$$

Therefore, for a node Q_i at distance i from the root of Γ it holds that $\deg_{\text{tot}}(Q_i(0,\mathbf{T})) \leq \ell$ and that

$$\deg_x(Q_i(x,\mathbf{T})) \leq \deg_x(Q(x,\mathbf{T})) + \ell i \leq (\ell+1)(k-1) + \ell k = \mathcal{O}\left(\ell k\right). \quad (5.38)$$

The number of monomials $x^i\mathbf{T}^{\mathbf{j}}$ for which $\deg_{\mathbf{w}}\left(x^i\mathbf{T}^{\mathbf{j}}\right) < \Delta$, that is

$$i + (k-1)\sum_{h=1}^{v} j_h < \Delta,$$

is at most $\Delta\left(\frac{\Delta}{k-1}\right)^v$. Therefore Equation (5.38) gives that Q_i has at most $\mathcal{O}\left(\ell^{v+1}k\right)$ terms. Algorithm 5 only performs computations in line 8 and 10, and we now check that the complexity of these computations is polynomial in $q^{\binom{\ell+v}{v}}$. To this end let P denote a node in Γ at distance i from the root.

- Above we saw that the total degree of $P(0,\mathbf{T})$ is at most ℓ. Hence $P(0,T,\dots,T)$ has degree at most ℓ, which means that in line 8, its roots can be computed deterministicly in a complexity which is polynomial in ℓ and q, either by the algorithm in [11] or simply by exhaustive search.

- In line 10 the polynomial P_{f_0} must be computed for up to q different values of f_0. By Equation (5.26) we can compute P_{f_0} by evaluating the Hasse–derivatives of P in the point $(0, f_0, \dots, f_0)$. The number of non-zero Hasse–derivatives is at most the number of terms in Q, and furthermore the complexity of computing a single Hasse–derivative is at most linear in this quantity. Hence all the Hasse–derivatives in Equation (5.26) can be computed in complexity $\mathcal{O}\left((\ell^{v+1}k)^2\right)$. The number of terms in a Hasse–derivative of P is at most the number of terms in P, and hence this quantity is $\mathcal{O}\left(\ell^{v+1}k\right)$. Using Horner's rule, it therefore follows that a Hasse–derivative of P can be evaluated in $(0, f_0, \dots, f_0)$ in complexity $\mathcal{O}\left(\ell^{v+1}k\right)$. Therefore the total complexity of line 10 is

$$\mathcal{O}\left((\ell^{v+1}k)^2 + q \cdot \ell^{v+1}k \cdot \ell^{v+1}k\right) = \mathcal{O}\left(\ell^{2(v+1)}q^3\right).$$

Using the above two items, we conclude that the complexity of the computations performed in a single node of Γ is polynomial in $q^{\binom{\ell+v}{v}}$. As argued above, this proves the proposition. $\square$

Remark 5.26. The result in Theorem 5.23 was discovered while preparing this thesis. Thus we have not been able to investigate any of its consequences, nor to make any serious attempts to weaken the assumption $v \leq 2$.

5.5.2 A bound on the list size under assumptions

In the previous section we found a bound on the size of the list computed by $\Lambda_k(Q)$ under the assumption $v \leq 2$. In this section we will derive a similar result without limitations on v, but under certain simplifying assumptions on the multiplicities of the zeroes of $Q(x, \mathbf{T})$. These assumptions are not in general satisfied by the polynomials computed in the interpolation step, and thus the results below should only be considered preliminary.

Proposition 5.27. *Let $Q(x, \mathbf{T}) \in \mathbb{F}_q[x, \mathbf{T}]$ be such that $Q(0, \mathbf{T}) \neq 0$, and let ℓ denote the total degree of $Q(0, \mathbf{T})$. Let $k < q - 1$, $v < q$ and assume that all zeroes in $Q(x, \mathbf{T})$ of the form $(0, f_0, \ldots, f_0) \in \mathbb{F}_q^{v+1}$ have multiplicity one. Then it holds that*

$$|\Lambda_k(Q)| \leq \ell q^{v-1}.$$

Proof. If $v = 1$ then the proposition follows from Theorem 5.23. In the following we therefore assume that $v \geq 2$. Let Γ denote the recursion tree of $\Lambda_k(Q)$, and let

$$Q = Q_0, \ldots, Q_d \tag{5.39}$$

be a branch in Γ beginning in the root node. Furthermore, for $1 \leq i \leq d$ let $f_{i-1} \in \mathbb{F}_q$ be a zero of $Q_{i-1}(0, T, \ldots, T)$ such that $Q_i = (Q_{i-1})_{f_{i-1}}$. See Figure 5.3 for an illustration of such a branch. The proposition will follow from Lemma 5.22 if we can show that in the branch in Equation (5.39) there can be at most $v - 1$ unconstrained nodes, and thus we aim to prove this.

Since $(0, \mathbf{f}_0) = (0, f_0, \ldots, f_0)$ is a zero in $Q_0(x, \mathbf{T})$ of multiplicity one, it follows from Lemma 5.18 that the total degree of $Q_1(0, \mathbf{T})$ is at most one. Therefore it follows from Corollary 5.19 that

$$\deg_{\text{tot}}(Q_i(0, \mathbf{T})) \leq \deg_{\text{tot}}(Q_1(0, \mathbf{T})) \leq 1,$$

for $i \geq 1$. Assume that for some node Q_i it holds that $\deg_{\text{tot}}(Q_i(0, \mathbf{T})) < 1$, then $Q_i(0, T, \ldots, T) = Q_i(0, \mathbf{T})$ is a constant polynomial. Furthermore, we have from Lemma 5.14 that $Q_i(0, \mathbf{T})$ is non-zero, and this implies that the branch terminates in the node Q_i, i.e. that $i = d$, and that Q_i is not unconstrained. Therefore, in the remainder of our argumentation we can assume that

$$\deg_{\text{tot}}(Q_i(0, \mathbf{T})) = 1, \tag{5.40}$$

for $1 \leq i \leq d$. We split the rest of the proof in two cases.

First, assume that $Q_0(0, T, \ldots, T) \neq 0$. Lemma 5.20 in combination with Equation (5.40) implies that among the nodes $Q_1, \ldots, Q_d$ there can be at most $\binom{1+v-1}{v-1} - 1 = v - 1$ unconstrained nodes. Since Q_0 is not unconstrained, this proves the lemma in this case.

Next, assume that $Q_0(0, T, \ldots, T) = 0$. In this case it follows from the chain rule that we have

$$0 = \mathcal{H}_T^{(1)}\left(Q_0(0, T, \ldots, T)\right) = \sum_{h=1}^{v} \mathcal{H}_{T_h}^{(1)}\left(Q_0\right)(0, T, \ldots, T),$$

and hence

$$\sum_{h=1}^{v} \mathcal{H}_{T_h}^{(1)}\left(Q_0\right)(0, \mathbf{f}_0) = 0. \tag{5.41}$$

By Equation (5.40) the branch $Q_1, \ldots, Q_d$ is exactly of the type considered in the proof of Lemma 5.20. Therefore we can argue in the same way as in that proof, to get that for $1 \leq i < d$, $(0, \mathbf{f}_i)$ is a zero of multiplicity one in $Q_i(0, \mathbf{T})$. Therefore, if we specialize Equation (5.31) to our current setting, we get that

$$\mathcal{H}_{T_h}^{(1)}\left(Q_i\right)(0, \mathbf{f}_i) = \alpha^{i(h-1)} \mathcal{H}_{T_h}^{(1)}\left(Q_0\right)(0, \mathbf{f}_0), \tag{5.42}$$

for $1 \leq i \leq d$ and $1 \leq h \leq v$. Now assume that along the branch $Q_1, \ldots, Q_d$, there are $v - 1$ indices $1 \leq i_1 < i_2 < \cdots < i_{v-1} \leq d$ such that $Q_{i_u}(0, T, \ldots, T) = 0$ for $1 \leq u \leq v - 1$. A necessary condition for the polynomial $Q_{i_u}(0, T, \ldots, T)$ to vanish, is that coefficients to $T_1, \ldots, T_v$ in $Q_{i_u}(0, \mathbf{T})$ sum to zero. By the equations (5.41) and (5.42) this means that the vector

$$\left(\mathcal{H}_{T_h}^{(1)}\left(Q\right)(0, \mathbf{f}_0)\right)_{h=1,\ldots,v} \in \mathbb{F}_q^v,$$

must be an element in the right kernel of the following matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \alpha^{i_1} & \cdots & \alpha^{i_1(v-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{i_{v-1}} & \cdots & \alpha^{i_{v-1}(v-1)} \end{bmatrix}.$$

Since $i_1 < i_2 < \ldots < i_{v-1} \leq k < q - 1$ and $v < q$, we have that $\mathbf{A}$ is a non-singular Vandermonde matrix. This implies that $\mathcal{H}_{T_h}^{(1)}\left(Q\right)(0, \mathbf{f}_0) = 0$, for $1 \leq h \leq v$. Since we assumed $(0, \mathbf{f}_0)$ to be a zero of multiplicity one in $Q(x, \mathbf{T})$

we therefore get that $\mathcal{H}_x^{(1)}(Q)(0,\mathbf{f}_0) \neq 0$. Furthermore, by Lemma 5.18 it follows that

$$Q_1(0,\mathbf{T}) = \mathcal{H}_x^{(1)}(Q)(0,\mathbf{f}_0) + \sum_{h=1}^{v} \alpha^{h-1}\mathcal{H}_{T_h}^{(1)}(Q)(0,\mathbf{f}_0)T_h = \mathcal{H}_x^{(1)}(Q)(0,\mathbf{f}_0),$$

and hence $Q_1(0,\mathbf{T})$ is a non-zero constant polynomial. This means that the branch $Q_1,\ldots,Q_d$ terminates at Q_1, i.e. that $d=1$, and also that Q_1 is not unconstrained. This contradicts our assumption of $v-1 \geq 1$ unconstrained nodes in the branch $Q_1,\ldots,Q_d$, and therefore we conclude that among $Q_1,\ldots,Q_d$, at most $v-2$ nodes are unconstrained. This means that in the full branch $Q_0,\ldots,Q_d$ there can be at most $v-1$ unconstrained nodes, and this proves the proposition. □

The polynomial $Q(x,T_1,T_2)$ from Example 5.15 is an element in the ideal $\langle x, T_1 - T_2\rangle^{r+1}$ and hence any zero in this polynomial of the form $(0,f_0,f_0)$, is in fact a zero of multiplicity at least $r+1$. Therefore, Proposition 5.27 does not apply to this polynomial, and in fact we saw in Example 5.15 that $|\Lambda_k(Q)| > \ell q$. Therefore the assumption in the above proposition that any zero of the form $(0,\mathbf{f}_0)$ in $Q(x,\mathbf{T})$ is of multiplicity one, can not be removed.

Remark 5.28. In algebraic-geometric language the polynomial Q_{f_0} defined in Equation (5.17) is called the *blow-up* of Q at the point $(0,f_0,\ldots,f_0)$ [32, p. 80]. The study of blow-ups is central in the problem of desingularization of varieties. For this reason the study of recursion trees undertaken in this section, bears many resemblances to the study of desingularization. For a thorough survey of the challenges of desingularization in positive characteristic see [33, 34].

5.6 Concluding remarks

At first glance it is not so transparent what it really is that makes the folding operation on the Reed–Solomon codes work so well. In [29] Guruswami and Rudra argue that the folding operation reduces the number of error-patterns that the (adversarial) channel may impose on the transmitted word, and since this too reduces the number of error-patterns that the list decoder has to “recognize”, the folding operation simplifies the list decoding task at the receiver end (at the expense of a larger code alphabet). In this light the real wonder of the folded Reed–Solomon codes is that one can actually find an algebraic method that takes advantage of this simplification.

In Section 5.4.1 we saw that it took an algebraic "coincidence" of two simultaneous properties of β, to get bounds on the list size and root-finding complexity. When the work covered in Section 5.4 was undertaken, the initial hope was that the Hensel–lifting approach, besides perhaps being more efficient, would also be more *flexible* in the sense that it would be independent of the special algebraic features of β. In particular it was the hope that the Hensel–lifting method would reveal more insight into what properties of the folding operation and the resulting root-relation

$$Q(x, f(x), f(\alpha x), \ldots, f(\alpha^{v-1}x)) = 0, \tag{5.43}$$

that are the keys to the optimal list decodability of the folded Reed–Solomon codes. Unfortunately it has not been possible to fully achieve this. However, we can extract some partial results from the previous sections:

- In Theorem 5.23 we found a list size bound that is polynomial in the code length, for the special case $v \leq 2$. Furthermore, in Proposition 5.27 we saw that under certain non-singularity assumptions on the interpolation polynomial, it is also possible to obtain polynomial list size bounds.

  We believe that the techniques used in the proofs of these results may be used to derive list size bounds under weaker assumptions and that this will be an interesting starting point for future work on the Hensel–lifting algorithm.

- In the proofs of Theorem 5.23 and Proposition 5.27 we crucially relied on the fact that the order of α (in the group $\mathbb{F}_q^*$) is *larger* than k. It seems that the large order of α is an important property of the root-relation (5.43).

- In Example 5.15 we saw that any general bound on the size of the list computed with the Hensel–lifting algorithm, must be strictly larger than ℓq^{v-1}.

We conclude with a remark on the alphabet size of folded Reed–Solomon codes. A disadvantage of folded Reed–Solomon codes is that their alphabets are very large for long codes. In [26] an attempt to deal with this fact is made. In the paper, the folding operation described in Section 5.1.2 is generalized to an operation on a special class of algebraic-geometry codes, defined over cyclotomic function fields with cyclic Galois groups. For a fixed alphabet size, the folded algebraic-geometry codes allow for longer codes, than folded Reed–Solomon codes. Thus the folded algebraic-geometry codes are less affected by the large alphabet-problem. However, the larger code length is achieved at cost of an extra layer of technicality, and since implementation of folded Reed–Solomon codes already seems a difficult task, practical realizations of folded algebraic-geometry codes remain an open challenge.

Chapter 6

Summary and discussion

In this thesis we have investigated the efficiency of list decoding algorithms based on the principles of the Guruswami–Sudan algorithm for two types of algebraic codes, namely simple C_{ab} codes and folded Reed–Solomon codes. Below we summarize these investigations and the main results are emphasized. Furthermore, points of interest for future investigations are given.

Key-equation: In Chapter 2 we formulated a general multivariate interpolation problem with degree constraints, encompassing all the interpolation problems encountered in various list decoders in later chapters. Furthermore, we reformulated the general interpolation problem as key-equations and showed that finding a valid interpolation polynomial is equivalent to finding a "short" vector in a module over a univariate polynomial ring. The generality of the approach allows for a number of conclusions:

- The derived key-equations contain the key-equations for various algebraic codes known in the literature, as special cases. Therefore the results in Chapter 2 may be taken as a *unified* view on key-equations. For instance, at first glance the key-equations of Lee and O'Sullivan [43] and those of Augot and Zeh [4], bears little resemblance. However, we saw that in the light of our derivations, they in fact only differ by a single rewriting.
- We saw that the re-encoding technique of Ahmed, Kötter, Ma and Vardy [39], can be naturally understood in the language of our key-equations.

In our derivations of the key-equations, it was necessary to make certain assumptions on the interpolation points. Although these assumptions seem to be an integral part of the derivations, it will be interesting to investigate whether key-equations can be infered under more relaxed assumptions.

The short-basis algorithm: In Chapter 3 we described the short-basis algorithm. This is a general algorithm which uses a divide–and–conquer approach to compute "short" bases of modules over a univariate polynomial ring, when the shortness is measured with respect to weighted degree. We saw that the algorithm can also be used to compute Gröbner bases of modules over a univariate polynomial ring with respect to term orders extending weighted degree.

We gave a detailed analysis of the short-basis algorithm and its asymptotic performance, and we found that it is highly efficient for modules defined by a basis of high weighted degree. This result is central to the developments in Chapter 4 since the interpolation algorithm described there, derives its efficiency directly from that of the the short-basis algorithm.

Multivariate polynomial interpolation: In Chapter 4 we gave an algorithm for solving the general interpolation problem formulated in Chapter 2. The algorithm builds on the key-equations derived in Chapter 2 and its key ingredient is the short-basis algorithm from Chapter 3. We applied the resulting interpolation algorithm to the interpolation step of the Guruswami–Sudan list decoding algorithm for simple C_{ab} codes. From this the following results are derived:

- An efficient algorithm for the interpolation step in the Guruswami–Sudan list decoding algorithm for Reed–Solomon codes, is obtained. The asymptotic complexity of this algorithm is

  $$\mathcal{O}\left(\ell^5 n \log^2(\ell n) \log\log(\ell n)\right).$$

  This compares favourably to other algorithms known in the literature.

- Furthermore, we obtained an efficient algorithm for the interpolation step in the Guruswami–Sudan list decoding algorithm for Hermitian codes. The asymptotic complexity of this algorithm is

  $$\mathcal{O}\left(\ell^5 n^2 \log^2(\ell n) \log\log(\ell n)\right),$$

  which is lower than the complexity of previously known algorithms.

6. Summary and discussion

We discussed the challenges of realizing the theoretical efficiency of the interpolation algorithm in practical implementations. From this discussion we conclude that:

- To get an efficient implementation of the interpolation algorithm it is crucial to have an efficient implementation of the short-basis algorithm.
- To obtain an efficient implementation of the short-basis algorithm it is necessary to use a hybrid of the short-basis algorithm and ordinary Gaussian elimination. In Section 4.2.3 we demonstrated that such a hybrid improves the practical performance of the short-basis algorithm significantly.
- To obtain an efficient implementation of the short-basis algorithm, a low-level implementation (e.g. in `C`) is necessary. The reason for this is the need to eliminate the excess time spent on memory and data-structure management in a high-level mathematical programming language.

By comparison of implementations in the high-level language `Magma` we saw that for large, but not unrealistic, problem instances our interpolation algorithm performs considerably better than the Lee–O'Sullivan interpolation algorithm [43]. Based on our experiments with these implementations, we conjecture that an implementation of the interpolation algorithm in which the above items are considered, will make it possible to achieve the efficiency promised by the asymptotic analysis. In particular we believe that the interpolation algorithm is *not* entirely theoretical, but really an algorithm with practical potential.

Folded Reed–Solomon codes: In Chapter 5 we described folded Reed–Solomon codes in detail. We showed that the codes can be list decoded with error-radius arbitrarily close to the optimal bound $1 - R$, for appropriately chosen parameters. Furthermore, we investigated the efficiency with which a list decoder for folded Reed–Solomon codes can be implemented. From these investigations we extract the following:

- We demonstrated that the interpolation algorithm from Chapter 4 can be applied to efficiently handle the interpolation step in the list decoder for folded Reed–Solomon codes. The complexity of the resulting algorithm is,

$$\mathcal{O}\left(\ell^{4v+1} Nm \log^2(\ell^{v+1} Nm) \log\log(\ell^{v+1} Nm)\right).$$

The dependence on the code length N of this complexity is significantly better than for the alternative linear-algebra approach. However, if $v > 1$ this comes at the cost of a worse dependence on ℓ and v.

- We described Guruswami and Rudra's approach to the root-finding step [27]. The advantage of this approach is that it allows for polynomial bounds (measured in the code length) on the running time of the resulting root-finding algorithm. However, although the running time is polynomial, we demonstrated that in practice the algorithm can be rather time-consuming.
- We presented an alternative algorithm for the root-finding step based on Hensel–lifting. It has not been possible to derive any general polynomial guarantees on the size of the list returned by this algorithm, nor on its running time. However, we managed to derive polynomial bounds under certain assumptions. More precisely we showed that if $v \leq 2$ or if all roots of the form $(0, \mathbf{f}_0)$ in the polynomial $Q(x, \mathbf{T})$ given as input to the algorithm are of multiplicity one, then the list size is polynomial. We believe that the techniques used in the derivations of these results provide a good starting point for future work on obtaining general bounds on the list size of the Hensel–lifting algorithm.
- Through simulations we demonstrated that, despite the lack of general guarantees on the Hensel–lifting algorithm's performance, it is faster in practice than Guruswami and Rudra's algorithm.

The current lack of general polynomial guarantees on the running time of the Hensel–lifting algorithm, makes it unsuitable for any theoretical applications in list decoding. Therefore, an interesting point for future investigations is to obtain such bounds.

The advantage of the Hensel–lifting approach is that it, besides being more efficient, also seems to be independent of the "algebraic coincidence" described in Section 5.4.1. The folded Reed–Solomon code's current reliance on these special algebraic features, makes it hard to generalize them. Moreover, this reliance even makes it hard to fully grasp precisely what properties of the folding operations are the keys to the folded codes optimal list decodability. A general bound on the running time of the Hensel–lifting algorithm (or more precisely its proof) will necessarily reveal insight into this question. This is another, and important, reason to invest further efforts into bounding the list size and running time of the Hensel–lifting algorithm.

Appendix A

Complexity analysis of Algorithm 3

In this appendix we derive two auxiliary results to facilitate the complexity analysis of of Algorithm 3 in Section 4.1. In Section A.1 we will analyse the complexity of computing the matrix $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ and in Section A.2 a bound on $\deg_{\tilde{\mathbf{w}}}(\mathbf{B}_{s,\ell}(\mathbf{R}, E))$ is derived. These two results are central in the proof of Theorem 4.2.

A.1 The complexity of computing $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$

In this section we analyse the complexity of setting up $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ in line 2 of Algorithm 3. Let $(\mathbf{P}, \mathbf{Y}, s, \ell, \mathbf{w}, \Delta)$ be the interpolation problem that Algorithm 3 receives as input. In the following we will let this input be fixed.

Recall that we assume that the interpolation problem, is defined over a simple C_{ab} curve $\mathcal{C}$. We will let all notation regarding $\mathcal{C}$ be as in Section 2.3, and in particular we will let $F = F(X_1, X_2)$ denote the polynomial defining the curve.

In Section 2.4.1 we saw that an element $A \in \mathcal{R}$ can be written uniquely as $A = \sum_{i=0}^{\gamma-1} a_i(x_1)x_2^i$. In the following we will refer to such a representation of A, as its *normal form modulo F*, or simply its normal form, and denote it by

$$A \bmod F = \sum_{i=0}^{\gamma-1} a_i(x_1)x_2^i.$$

We begin with a lemma showing how the vector $\mathbf{R}$ of Lagrange interpolation polynomials from Theorem 2.12, can be found efficiently.

A. Complexity analysis of Algorithm 3

Lemma A.1. *There exists* $\mathbf{R} \in \mathcal{R}^v$ *such that:*

- $R_j(P_i) = y_{ij}$ *for* $1 \leq i \leq n$ *and* $1 \leq j \leq v$.
- $\deg_{x_1}(R_j) < N$ *and* $\deg_{\mathbf{w}}(R_j) \leq \gamma(N+\delta)$ *for* $1 \leq j \leq v$.
- $\mathbf{R}$ *can be computed in time* $\mathcal{O}(vn(\gamma+N))$.

Proof. We use the notation of places in $\mathbb{F}_q(\mathcal{C})$ from Section 2.1.1. Thus we let $Q_1, \ldots, Q_N$ denote the places in S, and we let $P_{i,1}, \ldots, P_{i,\gamma}$ be the places in $\mathbb{F}_q(\mathcal{C})$ lying above Q_i. For each $1 \leq i \leq N$ let

$$L_i(x_1) = \prod_{\substack{h=1,\ldots,N \\ h\neq i}} (x_1 - x_1(Q_h)) \cdot \prod_{\substack{h=1,\ldots,N \\ h\neq i}} (x_1(Q_i) - x_1(Q_h))^{-1}.$$

Furthermore, for $1 \leq i \leq N$ and $1 \leq j \leq \gamma$ let

$$K_{i,j}(x_2) = \prod_{\substack{h=1,\ldots,\gamma \\ h\neq j}} (x_2 - x_2(P_{i,h})) \cdot \prod_{\substack{h=1,\ldots,\gamma \\ h\neq j}} (x_2(P_{i,j}) - x_2(P_{i,h}))^{-1}.$$

All the above polynomials do not depend on the received word, and hence we may assume them to be precomputed. We let the interpolation values in $\mathbf{Y}$ be indexed by the places $P_{i,j}$ for $1 \leq i \leq N$ and $1 \leq j \leq \gamma$, such that

$$\mathbf{Y} = \left(\mathbf{y}_{P_{1,1}}, \ldots, \mathbf{y}_{P_{1,\gamma}}; \cdots; \mathbf{y}_{P_{N,1}}, \ldots, \mathbf{y}_{P_{N,\gamma}}\right),$$

where each $\mathbf{y}_{P_{i,j}} = (y_{P_{i,j},1}, \ldots, y_{P_{i,j},v})$ is an element of $\mathbb{F}_q^v$. We can then get the desired $\mathbf{R} = (R_1, \ldots, R_v)$ by letting

$$R_b = \sum_{i=1}^{N} L_i(x_1) \left(\sum_{j=1}^{\gamma} y_{P_{i,j},b} K_{i,j}(x_2) \right). \tag{A.1}$$

Note that the weighted degree of R_b can be bounded by

$$\deg_{\mathbf{w}}(R) \leq \max\left\{\deg_{\mathbf{w}}(L_i K_{i,j}) \mid 1 \leq i \leq N, 1 \leq j \leq \gamma\right\} \leq \gamma N + \delta\gamma,$$

where the last equality follows since by Proposition 2.16, $-v_{P_\infty}(x_1) = \gamma$ and $-v_{P_\infty}(x_2) = \delta$. This proves the first claim of the lemma. Computing the polynomials (in x_2) in the parentheses in Equation (A.1) can be done by taking linear combinations of known polynomials, and thus the complexity of this is

$$\mathcal{O}\left(\sum_{i=1}^{N} \sum_{j=1}^{\gamma} \deg_{x_2} K_{i,j} \right) = \mathcal{O}\left(N\gamma^2\right) = \mathcal{O}(n\gamma).$$

Finally, to compute R_b we need to multiply each L_i into one of the parentheses in Equation (A.1). We have that L_i is a polynomial in $\mathbb{F}_q[x_1]$ and the polynomial in the parenthesis into which L_i is multiplied, is a polynomial in $\mathbb{F}_q[x_2]$. Therefore multiplying with a single L_i into a parenthesis, can be done in complexity

$$\mathcal{O}\left(\#\text{terms in parenthesis} \cdot \deg_{x_1} L_i\right) = \mathcal{O}\left(\gamma N\right).$$

Therefore the total complexity of computing R_b from the expression in Equation (A.1) is $\mathcal{O}\left(n\gamma + \gamma N^2\right) = \mathcal{O}\left(n(\gamma + N)\right)$. Hence $\mathbf{R}$ can be computed in complexity $\mathcal{O}\left(vn(\gamma + N)\right)$. □

We will also need the following lemma for bounding the complexity of computing a bivariate polynomial product and its normal form modulo F.

Lemma A.2. *Let A and B be polynomials in $\mathbb{F}_q[x_1, x_2]$ and let $\delta_A = \deg_{x_1} A$, $\delta_B = \deg_{x_1} B$, $\gamma_A = \deg_{x_2} A$ and $\gamma_B = \deg_{x_2} B$. The normal form of the product AB modulo F can be computed in time,*

$$\mathcal{O}\left((\gamma_A + \gamma_B - \gamma + 1)\gamma M(\delta_A + \delta_B + (\gamma_A + \gamma_B)\delta) + M((\delta_A + \delta_B)(\gamma_A + \gamma_B))\right).$$

Furthermore, the x_1-degree of the result is at most

$$\delta_A + \delta_B + (\gamma_A + \gamma_B - \gamma + 1)\delta.$$

Proof. By [20, p. 244] the product AB can be computed in complexity

$$\mathcal{O}\left(\mathrm{M}\left((\delta_A + \delta_B)(\gamma_A + \gamma_B)\right)\right). \tag{A.2}$$

Now, for each $\gamma - 1 \leq j \leq \gamma_A + \gamma_B$, let

$$C^{(j)}(x_1, x_2) = \sum_{i=0}^{j} c_i^{(j)}(x_1) x_2^i,$$

denote the polynomial of x_2-degree at most j, obtained by eliminating powers of x_2 higher than j in AB, using the relation $F(x_1, x_2) = 0$. With this notation $C^{(\gamma-1)}$ is the polynomial we are interested in computing. Note that $C^{(\gamma_A+\gamma_B)} = AB$ and that

$$\deg c_i^{(\gamma_A+\gamma_B)} \leq \delta_A + \delta_B, \tag{A.3}$$

for all i. We can write the polynomial defining the curve $\mathcal{C}$, as $F(x_1, x_2) = \sum_{i=0}^{\gamma} f_i(x_1)x_2^i$, and without loss of generality we may assume that $f_\gamma = 1$. To eliminate x_2^j in $C^{(j)}$, one computes

$$\begin{aligned} C^{(j-1)} &= C^{(j)}(x_1, x_2) - c_j^{(j)}(x_1)x_2^{j-\gamma}F(x_1, x_2) \\ &= \sum_{i=0}^{j-\gamma-1} c_i^{(j)}(x_1)x_2^i + \sum_{i=j-\gamma}^{j} \left(c_i^{(j)}(x_1) - c_j^{(j)}(x_1)f_{i-j+\gamma}(x_1)\right) x_2^i \\ &= \sum_{i=0}^{j-\gamma-1} c_i^{(j)}(x_1)x_2^i + \sum_{i=j-\gamma}^{j-1} \left(c_i^{(j)}(x_1) - c_j^{(j)}(x_1)f_{i-j+\gamma}(x_1)\right) x_2^i. \end{aligned} \tag{A.4}$$

From this expression it follows that

$$\deg c_i^{(j-1)} \le \max\{\deg c_i^{(j)}, \deg c_j^{(j)} + \delta\},$$

and by iterating this and using (A.3), we get

$$\deg c_i^{(j)} \le \delta_A + \delta_B + (\gamma_A + \gamma_B - j)\delta. \tag{A.5}$$

This proves the second claim of the lemma. From (A.5) it follows that the (univariate) polynomial products in (A.4) can be computed in time

$$\mathcal{O}\left(\sum_{i=j-\gamma}^{j-1} \mathrm{M}\left(\deg c_j^{(j)} + \delta\right)\right) = \mathcal{O}\left(\gamma \mathrm{M}\left(\delta_A + \delta_B + (\gamma_A + \gamma_B)\delta\right)\right),$$

and this is then also the complexity of computing $C^{(j-1)}$ from $C^{(j)}$. Hence $C^{(\gamma-1)}$ can be computed from $C^{(\gamma_A+\gamma_B)}$ in time

$$\mathcal{O}\left((\gamma_A + \gamma_B - \gamma)\gamma \mathrm{M}\left(\delta_A + \delta_B + (\gamma_A + \gamma_B)\delta\right)\right).$$

This proves the lemma. □

By definition of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$, its entries consist of products of powers of the polynomials $R_1, \dots, R_v$ and E (possibly multiplied by a binomial coefficient or by -1). With Lemma A.2 at hand we can now estimate the complexity of computing the normal forms of all the entries of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$.

Lemma A.3. *The complexity of computing the normal forms of $E^i\mathbf{R}^\mathbf{j}$ modulo F for all pairs $(i, \mathbf{j})$ such that $\mathbf{j} \in \Delta_\ell$ and $0 \le i \le \max(0, s - \sum_{h=1}^{v} j_h)$, is*

$$\mathcal{O}\left(\gamma^2 s \ell^v M(\ell(\gamma\delta + N))\right).$$

Furthermore,

$$\deg_{x_1}\left(E^i\mathbf{R}^\mathbf{j} \bmod F\right) \le iN + ((\gamma - 1)\delta + N)\sum_{h=1}^{v} j_h.$$

Proof. The polynomials $E, E^2, \ldots, E^s$ do not depend on the input, and hence they can be precomputed. By definition E is an element of $\mathbb{F}_q[x_1]$ and hence $E \bmod F = E$. Furthermore,

$$\deg_{x_1} E^i \leq iN. \tag{A.6}$$

Let $(i, \mathbf{j})$ be such that $0 \leq i \leq s$ and $i + \sum_{h=1}^{v} j_h \leq \ell$. Furthermore, assume that some index j_u is positive. Then the normal form of $E^i\mathbf{R}^{\mathbf{j}}$ modulo F can be computed from the normal form of $E^i\mathbf{R}^{\mathbf{j}}/R_u$, by first multiplying by R_u and then reducing modulo F. By Lemma A.2 it holds that

$$\begin{aligned}\deg_{x_1}\left(E^i\mathbf{R}^{\mathbf{j}} \bmod F\right) &\leq \deg_{x_1}\left(E^i\mathbf{R}^{\mathbf{j}}/R_u \bmod F\right) + \deg_{x_1} R_u + (\gamma-1)\delta \\ &\leq \deg_{x_1}\left(E^i\mathbf{R}^{\mathbf{j}}/R_u \bmod F\right) + N + (\gamma-1)\delta,\end{aligned}$$

and using this together with (A.6) we get

$$\deg_{x_1}\left(E^i\mathbf{R}^{\mathbf{j}} \bmod F\right) \leq iN + ((\gamma-1)\delta + N)\sum_{h=1}^{v} j_h \leq \ell((\gamma-1)\delta+N). \tag{A.7}$$

This proves the second claim of the lemma. To prove the first claim, note that by Lemma A.2 one can compute the normal form of $E^i\mathbf{R}^{\mathbf{j}}$ from that of $E^i\mathbf{R}^{\mathbf{j}}/R_u$ in time

$$\begin{aligned}\mathcal{O}\Big(\gamma^2 \mathrm{M}\left(\deg_{x_1}\left(E^i\mathbf{R}^{\mathbf{j}}/R_u \bmod F\right) + N + \gamma\delta\right) \\ +\mathrm{M}\left(\gamma(\deg_{x_1}\left(E^i\mathbf{R}^{\mathbf{j}}/R_u \bmod F\right) + N)\right)\Big).\end{aligned}$$

Using the last inequality in Equation (A.7) the above complexity can further be estimated by

$$\mathcal{O}\left(\gamma^2\mathrm{M}\left(\ell(\gamma\delta+N)\right) + \mathrm{M}\left(\gamma\ell(\gamma\delta+N)\right)\right) = \mathcal{O}\left(\gamma^2\mathrm{M}\left(\ell(\gamma\delta+N)\right)\right),$$

where the last equality follows since the function $\mathrm{M}(\cdot)$ is at most quadratic. Since $|\Delta_\ell| \leq \ell^v$ this means that all the normal forms of all the polynomials $E^i\mathbf{R}^{\mathbf{j}}$ for $\mathbf{j} \in \Delta_\ell$ and $i \leq \max(0, s - \sum_h j_h)$ can be computed in time at most

$$\mathcal{O}\left(\gamma^2 s \ell^v \mathrm{M}\left(\ell(\gamma\delta+N)\right)\right).$$

□

We are now ready to derive the desired bound on the complexity of computing $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$.

Lemma A.4. *The complexity of computing the matrix* $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ *from Definition 2.21, is*

$$\mathcal{O}\left(\ell^{2v}\gamma M\left((\ell(\gamma\delta + N) + \gamma^2\delta)\gamma\right)\right).$$

Proof. In the proof we will for notational simplicity let $\mathbf{A}$ denote the matrix $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$. We will split the computation of $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ in two steps. First we will compute the normal form of $[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}$ for all $\mathbf{j}', \mathbf{j} \in \Delta_\ell$. Next, for each pair $(\mathbf{j}', \mathbf{j})$ and for each $0 \leq i \leq \gamma - 1$ we will compute the normal form of $x_2^i \cdot [\mathbf{A}]_{\mathbf{j}',\mathbf{j}}$. The coefficients to x_2 in these normal forms will then, by definition be the sought entries in $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$. We consider each step separately below:

1. By definition of $\mathbf{A}$ its entries are all of the form $E^i\mathbf{R}^\mathbf{j}$ with $\mathbf{j} \in \Delta_\ell$ and $0 \leq i \leq \max(0, s - \sum_{h=1}^{v} j_h)$, possibly multiplied by a binomial coefficient or by -1. By Lemma A.3 this means that the normal forms of the entries of $\mathbf{A}$ can computed in complexity $\mathcal{O}\left(\gamma^2 s\ell^v \mathrm{M}\left(\ell(\gamma\delta + N)\right)\right)$.

2. For $1 \leq i \leq \gamma - 1$ the normal form of $x_2^i \cdot [\mathbf{A}]_{\mathbf{j}',\mathbf{j}}$ can be computed from that of $x_2^{i-1} \cdot [\mathbf{A}]_{\mathbf{j}',\mathbf{j}}$. By Lemma A.2 this can be accomplished in time
$$\mathcal{O}\left(\gamma\mathrm{M}\left(\deg_{x_1}(x_2^{i-1}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}) + \gamma\delta\right) + \mathrm{M}\left(\gamma \deg_{x_1}(x_2^{i-1}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}})\right)\right),$$
and furthermore it holds that
$$\deg_{x_1}(x_2^{i}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}) \leq \deg_{x_1}(x_2^{i-1}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}) + 1 + (\gamma - 1)\delta.$$
By iterating this and using Equation (A.7) it follows that
$$\begin{aligned}\deg_{x_1}(x_2^{i}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}) &\leq \deg_{x_1}([\mathbf{A}]_{\mathbf{j}',\mathbf{j}}) + i(1 + (\gamma - 1)\delta) \\ &\leq \ell((\gamma - 1)\delta + N) + i(1 + (\gamma - 1)\delta) \\ &= \mathcal{O}\left(\ell(\gamma\delta + N) + \gamma^2\delta\right), \qquad \text{(A.8)}\end{aligned}$$
and therefore the total complexity of computing all the normal forms $x_2^i[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}$ for $1 \leq i \leq \gamma - 1$ and $\mathbf{j}', \mathbf{j} \in \Delta_\ell$ from the normal forms of $[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}$ can be bounded by
$$\sum_{i=1}^{\gamma-1} \sum_{\mathbf{j}' \in \Delta_\ell} \sum_{\mathbf{j} \in \Delta_\ell} \mathcal{O}\left(\gamma\mathrm{M}\left(\deg_{x_1}(x_2^{i-1}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}}) + \gamma\delta\right) + \mathrm{M}\left(\deg_{x_1}(x_2^{i-1}[\mathbf{A}]_{\mathbf{j}',\mathbf{j}})\gamma\right)\right).$$
Using Equation (A.8) and the fact that $|\Delta_\ell| \leq \ell^v$ we can further bound this quantity by
$$\begin{aligned}&\mathcal{O}\left(\ell^{2v}\gamma\left[\gamma\mathrm{M}\left(\ell(\gamma\delta + N) + \gamma^2\delta\right) + \gamma\delta\right) + \mathrm{M}\left((\ell(\gamma\delta + N) + \gamma^2\delta)\gamma\right)\right]\right) \\ &\qquad = \mathcal{O}\left(\ell^{2v}\gamma\mathrm{M}\left((\ell(\gamma\delta + N) + \gamma^2\delta)\gamma\right)\right),\end{aligned}$$
where the last equality follows since by Equation (3.17) we have $\mathrm{M}(\gamma t) \geq \gamma\mathrm{M}(t)$.

Using the fact that $s \leq \ell$ together with Equation (3.17) it follows that the complexity of step 2 dominates the complexity of step 1. Therefore the total complexity of setting up $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ is

$$\mathcal{O}\left(\ell^{2v}\gamma \mathrm{M}\left((\ell(\gamma\delta + N) + \gamma^2\delta)\gamma\right)\right).$$

□

A.2 An estimate of $\deg_{\tilde{\mathbf{w}}}(\mathbf{B}_{s,\ell}(\mathbf{R}, E))$

In the proof of Theorem 4.2, a bound on $\deg_{\tilde{\mathbf{w}}}(\mathbf{B}_{s,\ell}(\mathbf{R}, E))$ is needed. In the following lemma we provide such a bound.

Lemma A.5. *Let notation be as in Theorem 4.2, and let $\tilde{\mathbf{w}} = \rho(\mathbf{w})$. Assume that $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$ has been set up using the vector $\mathbf{R}$ from Lemma A.1, so that $\deg_{\mathbf{w}}(R_j) \leq \gamma(N + \delta)$ for $1 \leq j \leq v$. Furthermore, assume that*

$$w_j \leq \deg_{\mathbf{w}}(R_j), \quad \textit{for } 1 \leq j \leq v.$$

Then it holds that

$$\deg_{\tilde{\mathbf{w}}}(\mathbf{B}_{s,\ell}(\mathbf{R}, E)) = \mathcal{O}\left(\ell^{v+1}\gamma^2(N + \delta)\right).$$

Proof. We begin the proof by bounding the weighted degree of a single column in $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$. From Proposition 2.24 and from the definition of $\mathbf{B}_{s,\ell}(\mathbf{R}, E)$, we have that the weighted degree of the $(i, \mathbf{j})$-th column of this matrix is

$$\begin{aligned}\deg_{\tilde{\mathbf{w}}}\left((\mathbf{B}_{s,\ell}(\mathbf{R}, E))_{(i,\mathbf{j})}\right) &= \deg_{\rho(\mathbf{w})}\left(\rho\left(x_2^i \cdot (\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}\right)\right)\\ &= \deg_{\mathbf{w}}\left(x_2^i \cdot (\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}\right). \qquad (A.9)\end{aligned}$$

By definition, the $\mathbf{j}'$-th entry of the $\mathbf{j}$-th column of $\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E)$ is equal to the coefficient to $\mathbf{T}^{\mathbf{j}'}$ in $E^{\max(0, s - \sum_{h=1}^v j_h)}(\mathbf{T} - \mathbf{R})^{\mathbf{j}}$. Therefore,

$$\deg_{\mathbf{w}}\left((\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}\right) = \deg_{\mathbf{w}}\left(E^{\max(0, s - \sum_{h=1}^v j_h)}(\mathbf{T} - \mathbf{R})^{\mathbf{j}}\right).$$

By assumption we have $w_j \leq \deg_{\mathbf{w}}(R_j)$, which implies that

$$\deg_{\mathbf{w}}\left((\mathbf{T} - \mathbf{R})^{\mathbf{j}}\right) = \sum_{h=1}^{v} j_h \deg_{\mathbf{w}}(R_h). \qquad (A.10)$$

Furthermore, by definition of E we have that $\deg_{\mathbf{w}}(E) = -v_{P_\infty}(E) = n$. Hence by Equation (A.9) we get

$$\begin{aligned}\deg_{\tilde{\mathbf{w}}}\left((\mathbf{B}_{s,\ell}(\mathbf{R},E))_{i,\mathbf{j}}\right) &= \deg_{\mathbf{w}}\left(x_2^i \cdot (\mathbf{A}_\ell(-\mathbf{R})\mathbf{D}_{s,\ell}(E))_{\mathbf{j}}\right)\\ &= i\delta + n\max\left(0, s - \sum_{h=1}^{v} j_h\right) + \sum_{h=1}^{v} j_h \deg_{\mathbf{w}}(R_h).\end{aligned}$$

Summing over all the columns of $\mathbf{B}_{s,\ell}(\mathbf{R},E)$ we therefore get,

$$\begin{aligned}\deg_{\tilde{\mathbf{w}}}(\mathbf{B}_{s,\ell}(\mathbf{R},E)) &= \sum_{i=0}^{\gamma-1}\sum_{\mathbf{j}\in\Delta_\ell} \deg_{\tilde{\mathbf{w}}}\left((\mathbf{B}_{s,\ell}(\mathbf{R},E))_{i,\mathbf{j}}\right)\\ &\le \gamma\ell^v\left(\gamma\delta + ns + \ell\gamma(N+\delta)\right)\\ &= \mathcal{O}\left(\ell^{v+1}\gamma^2(N+\delta)\right),\end{aligned}$$

where the inequality follows from the assumption $\deg_{\mathbf{w}}(R_j) \le \gamma(N+\delta)$ and the last equality follows since $ns \le \ell\gamma N$. This proves the lemma. $\square$

Bibliography

[1] Overview of Magma v2.9 features. `http://magma.maths.usyd.edu.au/magma/Features/node93.html`.

[2] M. Alekhnovich. Linear Diophantine equations over polynomials and soft decoding of Reed–Solomon codes. *IEEE Transactions on Information Theory*, 51(7), 2005.

[3] D. Augot and L. Pecquet. A Hensel lifting to replace factorization in list–decoding of algebraic–geometric and Reed–Solomon codes. *IEEE Transactions on Information Theory*, 46(7), November 2000.

[4] D. Augot and A. Zeh. On the Roth and Ruckenstein equations for the Guruswami-Sudan algorithm. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 2620–2624, 2008.

[5] P. Beelen and K. Brander. Key-equations for list decoding of Reed–Solomon codes and how to solve them. Accepted for publication in Journal of Symbolic Computation, November 2008.

[6] P. Beelen and K. Brander. Efficient list decoding of a class of algebraic-geometry codes. Submitted to Advances in Mathematics of Communications, November 2009.

[7] P. Beelen and K. Brander. *Gröbner Bases, Coding, and Cryptography*, chapter Decoding Folded Reed–Solomon Codes Using Hensel–Lifting, pages 389–394. Springer-Verlag, 2009.

[8] P. Beelen and T. Høholdt. List decoding using syndromes. In J. Chaumine, J. Hirschfeld, and R. Rolland, editors, *Algebraic Geometry and its Applications: Proceedings of the first SAGA conference*, pages 315–331, May 2007.

[9] E. Ben-Sasson, S. Kopparty, and J. Radhakrishnan. Subspace polynomials and list decoding of Reed–Solomon codes. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:207–216, 2006.

[10] E. R. Berlekamp. *Algebraic coding theory.* McGraw-Hill, 1968.

[11] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713–735, 1970.

[12] E. R. Berlekamp. Bounded distance+1 soft-decision Reed–Solomon decoding. *IEEE Transactions on Information Theory*, 42:704–720, 1996.

[13] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. the user language. *Journal of Symbolic Computation*, 3-4:235–265, 1997.

[14] K. Brander. Source code for Magma implementations of algorithms related to list decoding. `http://www2.mat.dtu.dk/people/K.Brander/`, 2010.

[15] Y. Cassuto and J. Bruck. A combinatorial bound on the list size. Technical Report ETR058, California Institute of Technology, May 2004.

[16] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry.* Graduate Texts in Mathematics. Springer, second edition, 2004.

[17] Z. Dvir, S. Kopparty, S. Saraf, and M. Sudan. Extensions to the Method of Multiplicities, with applications to Kakeya sets and mergers. *Electronic Colloquium on Computational Complexity*, (4), 2009.

[18] P. Elias. List decoding for noisy channels. Technical report, Research Laboratory of Electronics, MIT, 1957.

[19] G.-L. Feng and T. Rao. A simple approach for construction of algebraic-geometric codesfrom affine plane curves. *IEEE Transactions On Information Theory*, 40(4):1003–1012, July 1994.

[20] J. v. z. Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, second edition, 2003.

[21] O. Goldreich and L. A. Levin. A hard–core predicate for all one–way functions. In *STOC '89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, 1989.

[22] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: The highly noisy case. *SIAM Journal on Discrete Mathematics*, 13(4):535–570, November 2000.

[23] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science.* Addison–Wesley Professional, 2nd edition edition, 1994.

[24] V. Guruswami. Limits to list decodability of linear codes. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 802–811, 2002.

[25] V. Guruswami. Algorithmic results in list decoding. *Foundations and Trends in Theoretical Computer Science*, 2(2), 2007. Now Publishers.

[26] V. Guruswami. Artin automorphisms, cyclotomic function fields, and folded list-decodable codes. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 23–32, Bethesda, MD, USA, May 2009.

[27] V. Guruswami and A. Rudra. Explicit capacity-achieving list-decodable codes. *Electronic Colloquium on Computational Complexity*, Report no. 133, 2005.

[28] V. Guruswami and A. Rudra. Explicit capacity-achieving list-decodable codes. In *STOC 2006*, Seattle, Washington, USA, May 21–23 2006.

[29] V. Guruswami and A. Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions On Information Theory*, 54(1):135–150, 2008.

[30] V. Guruswami and M. Sudan. Improved decoding of Reed–Solomon and algebraic-geometric codes. *IEEE Transactions On Information Theory*, 45:1757–1767, 1999.

[31] V. Guruswami and M. Sudan. Extensions to the Johnson bound. `http://people.csail.mit.edu/madhu/papers/johnson.ps`, February 2001.

[32] J. Harris. *Algebraic Geometry*. Springer, 1992.

[33] H. Hauser. The Hironaka theorem on resolution of singularities (or: A proof that we always wanted to understand). *Bulletin of the American Mathematical Society*, 40(3):323–403, 2003.

[34] H. Hauser. Why the characteristic zero proof of resolution of singularities fails in positive characteristic. `http://homepage.univie.ac.at/herwig.hauser/Publications/Why_CharZero_jul09.pdf`, 2003.

[35] M.-D. Huang and A. K. Narayanan. Folded algebraic geometric codes from Galois extensions. *ArXiv CoRR*, `http://arxiv.org/abs/0901.1162`, January 2009.

[36] J. Justesen. Iterated decoding of modified product codes in optical networks. Presented at Information Theory and Applications Workshop, February 2009. `http://ita.ucsd.edu/workshop/09/files/paper/paper_122.pdf`.

[37] J. Justesen and T. Høholdt. Bounds on list decoding of mds codes. *IEEE Transactions on Information Theory*, 47(4):1604–1609, 2001.

[38] J. Justesen and T. Høholdt. *A course in algebraic coding theory.* EMS Textbooks in Mathematics. European Mathematical Society, 2003.

[39] R. Koetter, J. Ma, A. Vardy, and A. Ahmed. Efficient interpolation and factorization in algebraic soft-decision decoding of Reed–Solomon codes. In *Information Theory, 2003. Proceedings. IEEE International Symposium on*, pages 365–365, June-4 July 2003.

[40] V. Y. Krachkovsky. Reed-solomon codes for correcting phased error bursts. *Information Theory, IEEE Transactions on*, 49(11):2975–2984, Nov. 2003.

[41] K. Lee and M. E. O'Sullivan. Algebraic soft-decision decoding of hermitian codes. 2008.

[42] K. Lee and M. E. O'Sullivan. List decoding of Hermitian codes using Gröbner bases. *Journal of Symbolic Computation*, In Press, Corrected Proof, 2008.

[43] K. Lee and M. E. O'Sullivan. List decoding of Reed-Solomon codes from a Gröbner basis perspective. *Journal of Symbolic Computation*, 43(9):645–658, 2008.

[44] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 1997.

[45] J. Ma and A. Vardy. A complexity reducing transformation for the Lee–O'Sullivan interpolation algorithm. In *Proceedings of IEEE International Symposium on Information Theory*, 2007.

[46] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, 1977.

[47] S. Miura and N. Kamiya. Geometric-Goppa codes on some maximal curves and their minimum distance. In *Proceedings of 1993 IEEE Information Theory Workshop*, pages 85–86, Shizuoka, Japan, June 1993.

[48] R. R. Nielsen. *List decoding of linear block codes.* PhD thesis, Technical University of Denmark, September 2001.

[49] V. Olshevsky and M. A. Shokrollahi. A displacement approach to efficient decoding of algebraic-geometric codes. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 235–244, New York, NY, USA, 1999. ACM.

[50] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami–Sudan radius in polynomial time. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 285–294, Washington, DC, USA, 2005. IEEE Computer Society.

[51] R. Pellikaan and X.-W. Wu. List decoding of q-ary Reed-Muller codes. *IEEE Transactions on Information Theory*, 50(4):679–682, 2004.

[52] V. S. Pless and W. C. Huffman, editors. *Handbook of Coding Theory*, volume 1. North-Holland, 1998.

[53] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Soceity of Industrial and Applied Mathematics*, 8(2):300–304, June 1960.

[54] R. M. Roth and G. Ruckenstein. Efficient decoding of Reed–Solomon codes beyond half the minimum distance. *IEEE Transactions On Information Theory*, 46(1):246–257, 2000.

[55] A. Rudra and S. Uurtamo. Two theorems in list decoding. *ArXiv CoRR*, `http://arxiv.org/abs/1001.1781`, January 2010.

[56] S. Sakata. N–dimensional Berlekamp–Massey algorithm for multiple arrays and construction of multivariate polynomials with preassigned zeros. In T. Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-correcting Codes: Proc. AAECC-6*, pages 356–376, 1989.

[57] S. Sakata. On fast interpolation method for Guruswami–Sudan list decoding of one-point algebraic-geometry codes. In *AAECC*, volume 2227 of *LNCS*, pages 172–181, 2001.

[58] S. Sakata, Y. Numakami, and M. Fujisawa. A fast interpolation method for list decoding of RS and algebraic-geometric codes. In *Proceedings of IEEE International Symposium on Information Theory*, page 479, Sorrento, Italy, June 2000.

[59] J.-P. Serre. *A Course in Arithmetic.* Springer-Verlag, 1973.

[60] I. R. Shafarevich. *Basic Algebraic Geometry 1.* Springer, second edition, 1994. Varieties in Projective Space.

[61] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, October 1948.

[62] M. A. Shokrollahi and H. Wasserman. List decoding of algebraic–geometric codes. *IEEE Transactions on Information Theory*, 45(2), 1999.

[63] H. Stichtenoth. *Algebraic Function Fields and Codes.* Universitext. Springer, 1993.

[64] V. Strassen. Gaussian elimination is not optimal. *Numerische Matematik*, 13:354–356, 1969.

[65] M. Sudan. Decoding of Reed–Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[66] P. Trifonov. Efficient interpolation in the Guruswami–Sudan algorithm. `http://arxiv.org/abs/0812.4937v2`, June 2009.

[67] L. Welch and E. R. Berlekamp. Error correction for algebraic block codes. U.S. Patent 4 633 470.

[68] J. M. Wozencraft. List decoding. *Quarterly Progress Report*, Research Laboratory of Electronics, MIT, 1958.

[69] X.-W. Wu and P. H. Siegel. Efficient root-finding algorithm with application to list decoding of algebraic-geometric codes. *IEEE Transactions On Information Theory*, 47(6):2579–2587, September 2001.

List of Symbols

Curves

$\mathcal{C}$	Curve defined by $F(x_1, x_2) = 0$ over $\mathbb{F}_q$	14
E_i	The element $x_1 - x_1(Q_i)$ of $\mathcal{R}$	15
E	The element $\prod_{i=1}^{n} E_i$ of $\mathcal{R}$	15
F	Absolutely irreducible polynomial defining $\mathbb{F}_q(\mathcal{C})$	14
$\mathbb{F}_q$	Finite field with q elements	14
$\mathbb{F}_q(\mathcal{C})$	Function field of $\mathcal{C}$	14
N	The size of S	15
P_i	Rational place of $\mathbb{F}_q(\mathcal{C})$ different from P_∞	15
$P_{i,j}$	Rational place of $\mathbb{F}_q(\mathcal{C})$ lying above Q_i	15
P_∞	The unique common pole of x_1 and x_2	14
Q_i	Rational place of $\mathbb{F}_q(x_1)$	15
$\mathcal{R}$	Ring of functions in $\mathbb{F}_q(\mathcal{C})$ with poles only at P_∞	15
S	The set of places $Q_1, \ldots, Q_N$	15

Interpolation

$\mathbf{A}_\ell$	Function $\mathcal{R}^v \to \mathrm{Mat}_{m_\ell \times m_\ell}(\mathcal{R})$	20
$\mathbf{D}_{s,\ell}$	Function $\mathcal{R}^v \to \mathrm{Mat}_{m_\ell \times m_\ell}(\mathcal{R})$	20
$\deg_{\mathbf{w}}$	Weighted degree	18
Δ	Weighted degree bound	19
Δ_ℓ	The set $\{\mathbf{b} \in \mathbb{N}_0^v \mid b_1 + \cdots + b_v \leq \ell\}$	16
$\mathcal{H}_{\mathbf{T}}^{(\mathbf{b})}$	The $\mathbf{b}$-th Hasse–derivative with respect to $\mathbf{T}$	17
ℓ	Degree bound	17
m_ℓ	The size of Δ_ℓ	17
$\mathbf{P}$	Interpolation points	17
$\mathbf{R}$	Vector of Lagrange interpolation polynomials	22
s	Multiplicity parameter	17
v	Number of indeterminates in $\mathbb{F}_q[\mathbf{T}] = \mathbb{F}_q[T_1, \ldots, T_v]$	16

List of Symbols

$\mathbf{w}$	Weight vector	18
$\mathbf{Y}$	Interpolation values	17

Short-basis algorithm

$\mathbf{A}$	$m \times m$ matrix whose columns span M	44
$\mathrm{LC}_{\mathbf{w}}$	Leading coordinate with respect to $\deg_{\mathbf{w}}$	44
m	Dimension of the matrix $\mathbf{A}$	44
M	Module over $\mathbb{F}_q[x_1]$	43
$\mathrm{M}(\cdot)$	Complexity of multiplication in $\mathbb{F}_q[x_1]$	56
π_t	Accuracy t approximation	51
$\mathbf{U}_{\mathbf{w}}(\mathbf{A}, t)$	Invertible matrix	47

Simple C_{ab} codes

D	The divisor $\sum_{i=1}^{n} P_i$	65
G	The divisor μP_∞	65
μ	The degree of G	65

Folded Reed–Solomon codes

α	Primitive element of $\mathbb{F}_q$	81
Δ	Weighted degree bound	82
ℓ	Degree bound	82
m	Folding parameter	79
N	Code length	79
s	Multiplicity parameter	81
v	Interpolation parameter	80

Index