List-decoding of error-correcting codes

Johan Sebastian Rosenkilde Nielsen

Supervisors: Tom Høholdt and Peter Beelen

Department of Mathematics
Technical University of Denmark

7th of February 2010

Abstract

Recently, Wu proposed in [24] a new approach to list decoding Reed-Solomon codes, quite different from the Guruswami-Sudan algorithm and its derivatives [1,2,10,18]. It promises lower running-time and has shown new perspectives of the codes. Sadly, the impact of this idea has hitherto been minimal; the contribution is indisputable, so the reason might be found in the idea's sole exposition: a convoluted article, presenting and proving a long range of intricate results on a strict page limit.

In this thesis, we give a complete, self-contained and novel presentation of Wu's decoding algorithm. The key mechanics of the algorithm build on a combination of several previous decoders, and we begin with a full presentation of these and their background theory. We then derive Wu's algorithm itself, with a clear division between the results in general algebraic theory, and the application of these on the decoding problem.

Preface

This thesis was prepared at the Department of Mathematics at the Technical University of Denmark in the period of September 2009 till February 2010. It corresponds to 30 ECTS credits, and it was part of the requirements for acquiring an M.Sc. degree in engineering.

I would like to thank my supervisors Tom Høholdt and Peter Beelen for their excellent, encouraging and enthusiastic guidance during the project. Special thanks to Tom Høholdt for introducing me to the field of algebraic coding theory, and for convincing me to write my Master's thesis on it – I have not regretted that.

Thanks to my brother Christoffer Rosenkilde Nielsen for advice on exposition and thesis structuring. Thanks to Jonas Braband Jensen for many encouraging conversations about the project and about nothing at all, and for being understanding when I left him stranded at ITU on a whim. Thanks to the other students of Room 136 and to the people of the Department of Mathematics for friendly company and an inspiring atmosphere. Finally, thanks to my girlfriend Carola for being loving, supporting and understanding in my times of absent-mindedness; I know there have been many.

Johan Sebastian Rosenkilde Nielsen

This version corrects minor spelling issues compared to the original.

Contents

Chapter 1

Introduction

Mass-communication and networked digital systems are growing ever more pervasive. We rely constantly on the transfer of information via digital channels to be error-free; the impact of communication failures ranges from minor nuisances, as when the DVD disc won't play, to catastrophes, as when a vital control signal cannot reach the space shuttle.

This spurs a continued interest in error-correcting codes. Such codes protect the signal by trading some channel capacity and computational power for increased reliability. The field has existed since Shannon's seminal paper [19] in 1948, but the research in improving the trade-off is on-going.

The mathematically beautiful Reed-Solomon codes were discovered as early as 1960 [20], promising excellent error-correction and having an abundance of fascinating algebraic properties. When they were found, both the digital systems and the known algorithms were still too slow for practical application of the codes. The first major break-through came in 1969 with the efficient Berlekamp-Massey minimum-distance decoding algorithm [3, 17]. This decoder can retrieve the sent information, as long as the number of errors is less than half the minimum distance.

Not coincidentally, half the minimum distance is also the maximum error-capability for which the receiver can always find a unique best guess as to what the transmitted message was. Algorithms that cross this bound must incorporate into their mathematical foundation the possibility of a list of equally likely candidates; hence the name *list decoders*. It would take almost three decades before the first efficient Reed-Solomon list decoder was found: Madhu Sudan published in 1997 a ground-breaking paper describing an algorithm which could in polynomial time decode beyond half the minimum-distance for some parameters of Reed-Solomon codes [22]; only two years later, he generalised it with Guruswami to work for all parameters [10].

The advent of the Guruswami-Sudan algorithm reinvigorated the research in Reed-Solomon codes. The last ten years has seen numerous speed optimisations to parts of the algorithm, e.g. [1, 2, 18], but the overall concept and decoding capability has remained somewhat static.

Recently, a quite different – more involved – Reed-Solomon list decoder due to Wu was published [24]. It is based on an observation that though the Berlekamp-Massey decoding algorithm cannot correct more errors than half the minimum distance, it still exposes crucial information about the errors in such cases. It then uses a generalisation to the foundational idea of the Guruswami-Sudan algorithm to solve the emerging problem. This ingenious combination presents a fresh approach to breaking the minimum-distance barrier.

Sadly, the impact of this publication has hitherto been minimal, and to the best of this author's knowledge, no major publication to this day has built on its results. The contribution cannot be disputed, so the reason for this might be found in the only available source for the idea: a journal paper on a strict page limit, presenting and proving a long range of novel results containing intricate details. The clarity of the presentation is bound to suffer, impeding a deeper understanding for any reader.

That is exactly the motivation for this thesis. We give a full-length, self-contained presentation of the entire evolution of algebraic coding theory leading up to and including Wu's algorithm, with all necessary results and proofs, written in one consistent notation, specially tailored for this purpose.

We begin with the well-known background theory for and proof of the Berlekamp-Massey decoder, and next, the Guruswami-Sudan decoder. Equipped with this basis, we can present the necessary new theory for Wu's algorithm, which is then collected into the final decoding algorithm.

Embedded within Wu's article lie many new results in general algebra, and we have sought to extract these and present them disentangled from the application in coding theory. With this theory in hand, we can give a simpler and clearer proof of Wu's algorithm.

This presentation is made in an effort to expose as clearly as possible the algebraic properties of Wu's algorithm and the relation it has to previous work. The hope is that this will make the algorithm more accessible; not only for practical use but in particular for theoretical analysis.

For the remainder of this chapter, we will more precisely set the scene of error-correcting codes: first with a general description and idealisation of the problem we are looking at, followed by an introduction to Reed-Solomon codes. Readers familiar with the standard channel models can entirely skip the former, and those familiar with Reed-Solomon codes can skim through the latter, paying heed mostly to the introduced notation. At the end of the chapter we provide an overview of the remainder of the thesis.

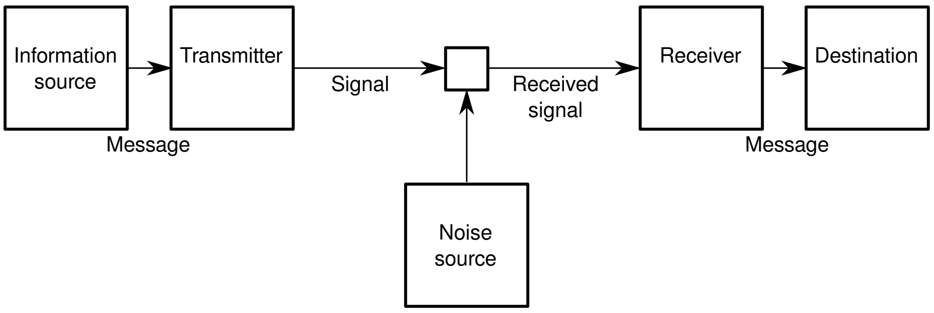

Figure 1.1: Shannon's idealised communication channel.

1.1 Channels and error-correcting codes

Imagine that someone wishes to send information to someone else via some channel: this could be a network-cable, satellite-communication or data encoded on a CD. Regardless of the medium, noise might occur: electrical surges, atmospheric radiation or scratches on the CD. This might prevent the receiver from retrieving the information correctly; depending on the type of information, an error in a single character might be enough to ruin the message.

Employing error-correcting codes is a way of adding redundancy to the sent message such that, if a few errors occur, the receiver can still retrieve the original information. Specifically, an error-correcting code is a description of how to *encode* information into a longer *codeword* and how to *decode* a possibly noised codeword to retrieve the original information. The complexity of encoding and decoding, the length of the codeword relative to the length of the original information, and the amount of noise the decoding can cater for, are all key factors deciding the quality of the error-correcting code. The decoding algorithm is usually the most mathematically challenging as well as computationally demanding.

Information theory was founded when this problem was first considered by Shannon in 1948 in [19]. He used the model in Figure 1.1 for the communication channels. When employing error-correcting codes, the transmitter is responsible for doing the encoding as well as the actual transmission over the channel. Likewise, the receiver is responsible for receiving the signal as well as performing the error-correction and decoding. In the diagram, the signal from one box to the next is perfect and error-free, except where the noise source is added. Shannon showed in the same publication, that for such channels, barring the case where the noise makes the received signal completely random, there always exists error-correcting codes that makes transmission possible with arbitrarily low error-probability. The amount of redundancy needed increases only steadily with the amount of noise on the channel, showing that we can always find "good" codes; the search for these promised codes continues to this day.

On top of this channel idealisation, many results in the literature – including those covered in this thesis – assume various other simplifications of the channels.

First of all, we will only consider digital channels; that is, channels where input and output is done on characters from a finite alphabet. For e.g. radio communication, this is a poor simplification, as read signals will often be somewhere between the digital characters, with some probability of being one or the other. Still, degrading a real-valued channel to a digital one – by converting in-between values to the most probable one – is often done in practice. Also, several error-correcting codes and decoding algorithms have been developed first for digital channels and then later generalised to handle real-valued channels better. As for the choice of alphabet, most modern computer systems work directly with binary, so this is a natural choice for the channels. However, in many settings of error-correction codes, the binary characters, or bits, are bunched together in *bytes* of some pre-defined length so as to enlarge the alphabet.

We will further limit the nature of the noise we regard: over the sent channel, each character has some probability p_{err} of being set to some – arbitrarily chosen – element of the alphabet. Each character is regarded independently and changed with the same probability. In particular, noise does not add or remove characters from the channel. Thus, the channels we will look at are particularly simple examples of so-called *discrete memoryless channels* [14]. Again, the realism of this modelling can be discussed. For e.g. a CD, a scratch will be contiguous, and it might also be circular as the layout of the bits; this will heighten the probability that a bit has noise if its neighbours has. In the case of CDs, the impact of this is limited by weaving different parts of the encoded stream together in order to increase the physical distance between neighbouring bits of the information.

Related to this, we will ignore the receiver's problem of finding the beginning and end of the message. With many channels, this is not a trivial problem; with e.g. satellite communication, atmospheric noise will constantly produce input for a satellite receiver, even when no messages are being sent. Still, we will assume that the receiver can find the transmitted message and its length, and furthermore, that he also knows the length of the information which he should attempt to retrieve.

Usually, the participants will beforehand agree on an *information word* length; the information to be sent will then be split up into words of this length, using some padding scheme for the last word. Each information word is then encoded and sent, as well as received and decoded individually. The length of each codeword is usually constant for all information words and also agreed upon beforehand. In such schemes, the ratio of each information word to encoding word is called the *rate* of the code; a simple measure for the throughput of the code.

The last important assumption is that the information to be sent is arbitrary. In many applications, this will not immediately be the case; e.g. when sending uncompressed images, video or sound, there might be enough structure and continuity that other application-specific error-correction methods will be more effective. The general assumption, though, is that any information word is equally likely to be sent, independently of previous or future information words.

The receiver therefore regards each received word individually, and his task is to recover the information word which was originally sent. This will always be a guess, and with no other information, the best strategy for the receiver is to assume the most probable event. For example, if the received word is a codeword, i.e. an encoded information word with no errors, there are two possibilities: either no errors occurred during transmission or enough errors occurred to transform the sent codeword into another codeword. If the channel is not extremely poor, i.e. as long as $p_{\text{err}} < \frac{1}{2}$, the former is more likely than the latter.

Let us give a simple example. Perhaps the most straight-forward error-correcting code is the 3-repetition code over the binary alphabet: encode 0 as 000 and 1 as 111. As each character of the information is then individually handled, we can let the information word length be 1; the codeword length is then 3 and the rate is $\frac{1}{3}$. The receiver then receives three binary characters, and, possibly, some of them were changed by noise. The best decoding algorithm is doing a vote between the received characters; if e.g 010 was received, it is more likely that one 0 was changed to 1, than it is for two 1's to have been changed to 0. This code is called *1-error correcting*: it decodes correctly up to 1 error in each codeword, and more errors are possibly decoded incorrectly (definitely, in this case) [14].

A decoding algorithm that always chooses the most probable sent information word is called a *maximum-likelihood decoder* [14]; for most practical error-correcting codes, the dream of finding such a decoding algorithm, which is also fast enough, is a proved impossibility. However, many algorithms behave as maximum-likelihood decoders for up to a certain number of errors; whenever more errors have occurred, they might report a fail or decode incorrectly. These are called *bounded-distance decoders* [14].

In some cases, the choice is not unique, and the decoding algorithm can choose between several, equally likely, information words; this arises when two or more codewords all have fewest characters different from the received word. We usually wish that a maximum-likelihood decoder should return a list of all equally likely information words in such a situation. Decoders supporting this are therefore often called *list decoders*.

This leads us to the concept of *minimum distance* of a code: the minimum number of pair-wise different characters between any two codewords of the code. Whenever there are fewer errors than half the minimum distance, there can be only the sent codeword closest to the received word, no matter which codeword was sent. Bounded-distance decoders bounded to half the minimum distance are called *minimum-distance decoders*, and they don't need to be list decoders. This fact seems to be reflected in the decoding algorithms: experience with practical codes shows us, that finding efficient minimum-distance decoders is quite easy; at least when compared to crossing the minimum-distance, where we know efficient list decoders for only a few families of codes.

In many cases, it is difficult to find the minimum distance or even good lower bounds on it. For others, e.g. the Reed-Solomon codes to be presented in the following section, we know it precisely. The minimum distance is often used as a simplistic way of comparing the efficacy of codes.

This concludes the general introduction and modelling of the error-correcting codes problem, and for the remainder of the thesis, we will concern ourselves with it only in the idealised setting just described. In the following section, we will formally model the elements of the communication in the setting of Reed-Solomon codes, and from then on, encoding and decoding will be analysed simply as relations between these mathematical objects.

1.2 Reed-Solomon codes

In this thesis, we will only consider the so-called Reed-Solomon (RS) codes; a large, efficient, and therefore also famous, family of algebraic codes.

To set up, the participants agree on the information word length k, the codeword length n as well as a finite field $\mathbb{F}_q$. The alphabet of the channel will be the q elements of $\mathbb{F}_q$; we therefore immediately have a plethora of operations, transformations and results on the elements of the alphabet to draw from. It is required that $n < q$, as well as of course $k < n$. The participants have also agreed upon n different elements of $x_0, x_1, \ldots, x_{n-1} \in \mathbb{F}_q$.

The sender now wishes to send an information word $w \in \mathbb{F}_q^k$ to the receiver. If we let $w = (w_0, w_1, \ldots, w_{k-1})$, we can regard the induced polynomial $w(x) = w_0 + w_1 x + \ldots + w_{k-1}x^{k-1}$; polynomials with coefficients from $\mathbb{F}_q$ form a polynomial ring $\mathbb{F}_q[x]$ so this makes sense. Encoding w to a codeword is now done as evaluation of $w(x)$ in the n points:

$$c = \big(w(x_0), w(x_1), \ldots, w(x_{n-1})\big) \tag{1.2.1}$$

During transmission, errors might occur, and the receiver receives some word r, also of length n. As the elements of both c and r are all in $\mathbb{F}_q$, we can see c and r as elements of the vector space $\mathbb{F}_q^n$. Now, as any element of a finite field can be sent into any other element by addition with some third element, we can regard the noise on the transmission as a vector $e \in \mathbb{F}_q^n$ such that $r = c + e$. If we let the number of errors be ϵ, then this is the number of non-zero elements of e.

As a small aside, let us immediately label this important measure as according to the literature: the number of non-zero elements of a vector v is its *Hamming weight*, or simply weight. Similarly, for two vectors of the same space v and w, the number of positions in which they differ is known as the *Hamming distance*, or simply distance, between them. Thus, ϵ is the weight of e and also the distance between r and c.

Some decoding algorithms perform decoding by finding w directly from r, while others first find e and then c. From c, it is easy to retrieve w by an operation similar to inverse discrete Fourier transformation; see for example [4].

As for w, we will often refer to the individual elements of c, r and e by adding an index to the name; that is e.g. $e = (e_0, \ldots, e_{n-1})$.

As described in the previous section, the minimum distance d is an important measure of a code, and it turns out that RS codes are excellent in this respect. Any RS code is *linear*, which means that the sum of two codewords, viewed as vectors, is again a codeword[1]. It is also easy to see, that for any linear code over a finite field, the minimum distance must be equal to the lowest weight of any non-zero codeword. This enables us to find a lower bound on the minimum distance: every zero in a codeword is due to a zero of the polynomial induced by the corresponding information word; as this polynomial is of degree at most $k-1$, and as $\mathbb{F}_q^n[x]$ is a polynomial ring, it can have no more than $k-1$ zeroes. Therefore, there must be at least $n-k+1$ non-zero elements. Thus, $d \geq n-k+1$. An important upper bound for *any* linear code, called the Singularity bound [14], states that $d \leq n-k+1$, and we therefore have equality. Linear codes satisfying this equality are often called *maximum distance separable* – MDS for short [16].

Minimum-distance decoders are the decoders that finds the unique closest codeword whenever the errors are fewer than half the minimum distance. If we should decoder further, we run the risk that several codewords are equally close to the received word, and hence equally likely to have been the sent word. The value $t = \lceil \frac{d}{2} \rceil - 1 = \lfloor \frac{n-k}{2} \rfloor$ there seem significant: the greatest integer less than half the minimum distance, and thus, the decoding bound for a minimum-distance decoder.

1.2.1 The considered subset

To simplify the exposition, we will only work with a restricted subset of the general RS codes. These have some additional nice properties, and in general, many of the equations and proofs regarding them are simpler. However, all of the algorithms presented here do in fact hold with the general codes in a related form.

First of all, we will always choose $n = q-1$. As finite fields only exist in certain sizes, this limits the codeword length considerably. However, even in many real-life applications, n is chosen as such for other practical concerns.

Secondly, we will always choose the n evaluation points as

$$(x_0, x_1, \ldots, x_{n-1}) = (\alpha^0, \alpha^1, \ldots, \alpha^{n-1}),$$

[1]This can be seen easily by using (1.2.1), remembering that all polynomials in $\mathbb{F}_q^n[x]$ of degree less than k correspond to an information word.

where α is a primitive element[2] of $\mathbb{F}_q$. These are all the elements of $\mathbb{F}_q \setminus \{0\}$ in a convenient order. Avoiding 0 is not necessary but does simplify some of the proofs.

These choices mean that encoding can be seen as a type of shifted discrete Fourier transform [4]. Retrieving w from c is easily done with the inverse of this operation.

For the remainder of the thesis, we will refer to all the values n, k, w, c, r, e, ϵ, t as well as the field $\mathbb{F}_q$ as they have been defined here.

1.2.2 Current state of affairs

For RS codes, very efficient minimum-distance decoders have been known for a long time, and these are the most commonly used, as speed is often a critical factor. The most famous of these are Berlekamp-Massey and the Sugiyama decoders.

In case $\epsilon > t$, there might be several codewords equally close to r, and a τ bounded-distance decoder with $\tau > t$ would have to be a list decoder. The first Reed-Solomon list decoders were the Sudan and Guruswami-Sudan algorithms [10, 22], which were found almost 40 years after the invention of the codes. Most other list decoders are variants and improvements on the Guruswami-Sudan algorithm. Its running time is asymptotically polynomial in n – and therefore "good" – but itself and all its derivatives are still much slower than the minimum-distance decoders. For this reason, the industry has yet to adopt them to any large extent.

The Sudan algorithm's decoding capability exceeds half the minimum distance only for codes of rate lower than $\frac{1}{3}$. The Guruswami-Sudan is a direct generalisation, which improves the capability all rates (for high enough n and k); the capability is a parameter of the algorithm, but it is limited to being less than $J = n - \sqrt{n(k-1)}$. The Guruswami-Sudan algorithm was mostly a proof-of-concept, its running time much too high for practical use; the last decade has seen numerous improvements to various parts of the algorithm, e.g. [1, 2], but few changes to the main concept of the algorithm.

Recently, Wu found a different list-decoding algorithm which is an ingenious combination of a generalised Guruswami-Sudan and the Berlekamp-Massey algorithms [24]. It promises lower running time than the Guruswami-Sudan and any of its variants, but it is uncertain as to which degree in practice. The τ parameter of the algorithm is noticeably again bounded by J, so its decoding capability is not better than the Guruswami-Sudan. However, Wu's algorithm has the huge advantage that whenever $\epsilon \leq t$, decoding is as fast as for the Berlekamp-Massey decoding algorithm.

It might therefore seem, that the expression J is significant when discussion decoding capability. In list-decoding settings, it is often (confusingly) known as the Johnson bound, as it is closely related the original Johnson bound: an upper bound on the

[2]That is, α multiplicatively generates $\mathbb{F}_q$: all α^i are different for $i = 0, \ldots, q-2$. As with all other elements of $\mathbb{F}_q$, we have $\alpha^{q-1} = 1$

size of constant-weight codes [13]. Using this original Johnson bound, it has been shown that for any received word, there are at most in the order of $O(n^2)$ many codewords within distance J [5, 11]; this result even holds for any code, linear or not[3]. This bodes well for the optimal efficiency on list decoders with decoding capability up to J.

Across the Johnson bound is still largely uncharted territory. It has been shown that in a setting slightly generalising that of decoding Reed-Solomon codes, crossing the Johnson bound will sometimes result in super-polynomially many codewords, precluding a polynomial time list-decoding algorithm [8]. In [18], however, Muralidhara and Sen show that for general RS codes, if we exceed J by a small amount, there will still only be polynomially many codewords. They also give a generalisation of the Guruswami-Sudan algorithm which decodes to this extent; however, its running time is a high-degree polynomial.

It is not too hard to see that for a polynomial time algorithm for *any* type of code, a certain upper bound on the decoding capability is $n - k$; informally, this is exactly the amount of redundancy added, and if there is more noise than redundancy, the possible number of codewords will explode. It is therefore impressive and intriguing that recently, Guruswami and Rudra in [9] show decoding algorithms for their so-called *Folded Reed-Solomon codes*, that achieves decoding capability arbitrarily close to $n - k$ for any rate. The price is a very large alphabet and high-degree polynomial running time. Their constructed codes are built upon RS codes, so their results might inspire improvements on decoding capability for these codes.

1.3 Overview

With a brief but solid foundation in coding theory with Reed-Solomon codes, we are now ready to commence, and we will here give an overview of what is in store. The thesis sets out to present, in full, Wu's list decoding algorithm. In order to do that, we need to prove and present both the theory and the previous decoders it is based upon.

In Chapter 2, we begin with well-known minimum-distance decoders; the purpose of the chapter is to arrive at the Berlekamp-Massey decoding algorithm and its background theory on linear recursions, but in the process, we will also meet the Sugiyama decoder.

After that, in Chapter 3, we will shift to an entirely different approach to decoding and present first the Sudan algorithm, and then the generalised Guruswami-Sudan algorithm.

[3]The general expression for the Johnson bound for any code is $\tilde{J} = n(1 - \sqrt{1 - \frac{d}{n}})$, which equals J in the case of RS codes.

We then get to the chapter on Wu's algorithm. Even though the algorithm is founded on the Berlekamp-Massey and the Guruswami-Sudan decoders, we will need quite a body of improvements to the background-theory on these, which we present in the first few sections of the chapter. We then assemble and prove the complete algorithm in Section 4.5, and then discuss various parts of the derivation and possibilities of improvements in Section 4.6.

In Chapter 5, we then summarise and conclude on the thesis.

For the reader, we provide a list of symbols in Appendix A, which contains a description on most of the "globally defined" symbols and variables used throughout the thesis; these are the ones that we might use far from their original definition.

CHAPTER 2

Minimum-distance decoding

For Reed-Solomon codes, minimum-distance decoders – decoders for up to half the minimum distance – have been known since the conception of the codes in 1960 [20]. Algebraic properties that allowed for much faster error correction was eight years later discovered and led to the Berlekamp-Massey decoding algorithm [3, 17]. Even by today's standards, the unmodified version of this algorithm remains an efficient and simple decoder.

The purpose of this chapter is to introduce the theory needed to understand the Berlekamp-Massey decoder using modern notation. We will not take any short-cuts, for already in the derivation of Wu's algorithm in Chapter 4, we will need all the details of the machinery.

We will begin by introducing two basic concepts: the syndromes in Section 2.1 and the error-locator polynomial in Section 2.2. These are derived values of e, and in the properties of these many decoding algorithms lie hidden. As an appetiser before the main presentation of the Berlekamp-Massey decoder, we show the Sugiyama decoding algorithm in Section 2.3; the short and elegant proof of this algorithm exemplifies well the beauty of the Reed-Solomon codes and the interaction between the introduced mathematical objects.

The core of the Berlekamp-Massey decoder can be seen as a method for solving *linear recursions*, and we will prove the correctness of the decoding algorithm using this approach. To do this, we present first a bulk of general algebraic theory on linear recursions in Section 2.4. We then return to coding theory and finally prove the Berlekamp-Massey decoder in Section 2.5.

2.1 Syndromes and the error spectrum

We will now introduce the basic concept of *syndromes*. These are essentially values which can be calculated from r by the receiver, but which depend only on the errors e. This makes them obvious targets for analysis, and many decoding algorithms also stem from the use of the syndromes.

For a linear code, a parity-check matrix H is a matrix such that for all codewords c, we have $Hc^{\mathrm{T}} = \underline{0}$; see for example [14]. For a received word r, we can then look at the vector

$$S = Hr^{\mathrm{T}} = H(c+e)^{\mathrm{T}} = He^{\mathrm{T}} \tag{2.1.1}$$

This is the definition of the syndromes. There is a general confusion of standard notation, as some use the word *syndrome* for the entire vector S, while others use it for the elements of the vector. We will do the latter, as we always look at only one received word r at a time.

Now, for any linear code, there are many parity-check matrices, but for our narrow-sense RS codes, it is well known that the following nice matrix is a parity-check matrix; see for example [14]:

$$H = \begin{bmatrix} 1 & \alpha & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \cdots & (\alpha^2)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k} & \cdots & (\alpha^{n-k})^{n-1} \end{bmatrix},$$

We will use this, so we define the syndromes of r as $S = (S_1, S_2, \ldots, S_{n-k}) = Hr^{\mathrm{T}}$. For $i = 1, \ldots, n-k$, we therefore have

$$\begin{aligned} S_i &= (1, \alpha^i, \ldots, (\alpha^i)^{n-1}) \cdot r \\ &= r_0 + \alpha^i r + \ldots + (\alpha^i)^{n-1} r_{n-1} \\ &= r(\alpha^i) \end{aligned}$$

Thus, the receiver can directly and easily compute the syndromes corresponding to the received word. Furthermore, from (2.1.1), we also have

$$S_i = (1, \alpha^i, \ldots, (\alpha^i)^{n-1}) \cdot e = e(\alpha^i), \quad i = 1, \ldots, n-k$$

For codewords, all the syndromes will be zero, and this also works the other way around. Therefore, a simple way of checking whether a word is a codeword, is to calculate all the syndromes, and see whether they are all zero.

It is natural to look at the evaluation of e at higher powers of α, so we define the *error spectrum* as

$$E = (E_1, \ldots, E_n), \qquad \text{where } E_i = e(\alpha^i)$$

Of course, we have $E_i = S_i$ for $1 \leq i \leq n-k$. Now, the row $h_j = (1, \alpha^j, \dots, (\alpha^j)^{n-1})$ is not parity-check row for $j > n-k$, so there, we do not necessarily have $h_j r^{\mathrm{T}} = h_j e^{\mathrm{T}} = E_j$; therefore, the receiver cannot directly compute the complete error spectrum. However, as we will see, it turns out that some of its properties allow for decoders based on the syndromes.

Notice that $E_n = e(\alpha^n) = e(\alpha^0)$, and one can see that $\tilde{E} = (E_0, E_1, \dots, E_{n-1})$ is the discrete Fourier transform of e over α. Therefore, if we had all the E_i, we could take the inverse discrete Fourier transformation of $\tilde{E}$ to obtain e. This demonstrates that all the information of e is embedded in E and vice versa.

Looking at E as a polynomial $E(x) = \sum_{i=0}^{n-1} E_{i+1} x^i$, we have a result which makes the above note precise. It is actually the less specific corollary following the proposition that we will use for decoding.

Proposition 2.1.1 $E(\alpha^{-i}) = -e_i \alpha^i$

Proof As hinted to, this can be seen using Fourier transformations, but can also be seen easily by direct calculation:

$$E(\alpha^{-i}) = \sum_{j=1}^{n} \alpha^{-i(j-1)} e(\alpha^j) \;=\; \alpha^i \sum_{j=1}^{n} \alpha^{-ij} \sum_{h=0}^{n-1} \alpha^{hj} e_h \;=\; \alpha^i \sum_{h=0}^{n-1} e_h \sum_{j=1}^{n} \alpha^{(h-i)j}$$

We see that the inner sum is a geometric series over α^{h-i}, so when $i \neq h$, we have

$$\sum_{j=1}^{n} \alpha^{(h-i)j} = \alpha \left(\frac{(\alpha^{h-i})^n - 1}{\alpha^{h-i} - 1} \right) \;=\; \alpha \left(\frac{1-1}{\alpha^{h-i} - 1} \right) = 0$$

Therefore, we continue

$$E(\alpha^{-i}) = \alpha^i \sum_{h=0}^{n-1} e_h \sum_{j=1}^{n} \alpha^{(h-i)j} \;=\; \alpha^i e_i \sum_{j=1}^{n} \alpha^{0j} \;=\; \alpha^i e_i \sum_{j=1}^{q-1} 1 \;=\; -e_i \alpha^i,$$

which concludes the proof. □

Corollary 2.1.2 *$E(x)$ has a zero at α^{-i} if and only if $e_i = 0$*

2.2 Error-locator polynomial

We will introduce yet another important derived value of e, which crops up in many decoders, and in particular the Berlekamp-Massey decoder.

We remember that ϵ is the number of errors and therefore the number of non-zero elements of e. Let now $0 \leq \ell_1 < \ell_2 < \ldots < \ell_\epsilon \leq n-1$ be the error positions – that is, exactly the i for which e_i is non-zero – and consider the polynomial

$$\Lambda(x) = \prod_{i=1}^{\epsilon}(1 - \alpha^{\ell_i} x)$$

It is easy to see that if j is an error-position, then $\Lambda(\alpha^{-j}) = 0$; furthermore, this specifies all the ϵ possible zeroes of the ϵ-degree polynomial. Therefore, the zeroes of this polynomial locates the errors; it is therefore aptly named the *error-locator polynomial.* Obviously, any polynomial thus "locating" the errors must have degree at least ϵ, and so, $\Lambda(x)$ is uniquely determined as having this property, degree ϵ and constant term 1.

This definition might seem unintuitive and farfetched; the receiver can't even calculate it as it depends on crucial information about e. However, as we will see, it turns out that it relates in many ways to the syndromes, and can be found by inspecting these.

Once $\Lambda(x)$ is found, we have efficient ways of obtaining e. First, we find the error positions by finding the roots of $\Lambda(x)$. In a finite field, this can be done by simply trying all elements of the field; this can be implemented a bit more efficiently – especially in hardware – using the Chien search [6].

With the error positions in hand, there are several related formulas for finding the error values; some of them depend on more information, usually retrieved as a bi-product of the method chosen for finding $\Lambda(x)$. The most general and famous is Forney's algorithm; it is out of the scope of this thesis to present Forney's algorithm, but it is quite simple and has low running time; see for example [4].

We will now present the key property of $\Lambda(x)$, which is foundational in many decoding algorithms. It is natural to suspect that E and $\Lambda(x)$ have some connection, but the result is still surprisingly elegant:

Theorem 2.2.1 *$\Lambda(x)$ is the lowest-degree non-zero polynomial solving for $p(x)$ the equation $p(x)E(x) \equiv 0 \mod x^n - 1$.*

Proof First we prove that $\Lambda(x)$ does indeed satisfy the equation: From Corollary 2.1.2, we know that $E(x)$ has a zero at α^{-i} if $e_i = 0$, and by definition, $\Lambda(x)$ has zero for all other powers of α. Therefore $\prod_{i=0}^{n-1}(x - \alpha^{-i}) \mid \Lambda(x)E(x)$, but $n = q-1$ and in $\mathbb{F}_q$ we have $\prod_{i=0}^{q-2}(x - \alpha^{-i}) = \prod_{i=0}^{q-2}(x - \alpha^{i}) = x^{q-1} - 1$, which gives the sought result.

Now we prove that any non-zero polynomial $p(x)$ satisfying the equation must have degree at least ϵ, so assume

$$p(x)E(x) \equiv 0 \mod x^n - 1,$$

As $x^n - 1 = \prod_{i=0}^{n-1}(x - \alpha^i)$, the above means that $\prod_{i=0}^{n-1}(x-\alpha^i) \mid p(x)E(x)$. From the form of $E(x)$ and Corollary 2.1.2, we know that only the terms $(x - \alpha^{-i}) = (x - \alpha^{n-i})$ where $e_i = 0$ divide $E(x)$. That leaves ϵ terms that must divide $p(x)$. $\square$

Let now Λ_i be the coefficient of the i'th term of $\Lambda(x)$ for all $i = 0, \ldots, \epsilon$. Note that $\Lambda_0 = 1$. We can then specialise the above theorem to using the syndromes; we have

$$\Lambda(x)E(x) \equiv 0 \mod x^n - 1 \iff \Lambda(x)E(x) = q(x)x^n - q(x),$$

for some polynomial $q(x)$ with $\deg(q(x)) < \epsilon$. This means that all coefficients of $\Lambda(x)E(x)$ for the terms of degree $\epsilon, \ldots, n-1$ must be zero. If we look at the expression for each coefficient of the product, this can be written as

$$\sum_{j=0}^{\epsilon} \Lambda_j E_{i-j} = 0, \quad i = \epsilon + 1, \ldots, n$$

If only a few errors occur – whenever $\epsilon < n - k$, to be precise – we can therefore write

$$\sum_{j=0}^{\epsilon} \Lambda_j S_{i-j} = 0, \quad i = \epsilon + 1, \ldots, n - k \tag{2.2.1}$$

We now suddenly have an equation linking the known S to the unknown $\Lambda(x)$, which looks very promising. This equation is also key in both the Sugiyama and the Berlekamp-Massey decoding algorithms.

We are now equipped with the most important values derived from e, and as hinted to by the above equation, a closer inspection of these can lead to decoding algorithms.

2.3 The Sugiyama algorithm

This section contains a presentation of the famous Sugiyama algorithm; an algebraically beautiful, simple, yet efficient decoder based on the classic Euclidean algorithm. It is not needed for understanding Wu's algorithm, so for the reader in a hurry – or one already familiar with the Sugiyama algorithm – this entire section can be skipped; we will, though, in the presentation of the Berlekamp-Massey decoder refer to one result proved here, namely Lemma 2.3.1.

The Sugiyama algorithm was discovered by Sugiyama et al. in 1975 [23], seven years later than the Berlekamp-Massey algorithm. These two algorithms have been shown to be in some sense essentially the same [12]; however, on the face of it, their respective background theory and derivation differ widely. This, of course, makes it all the more interesting, as there might be a deeper understanding hidden in their dissimilarities.

The Sugiyama algorithm stems from Theorem 2.2.1, and more specifically, the equation (2.2.1). We first wish to remove the unknown ϵ from that equation. In order to do that, we need to show a small lemma:

Lemma 2.3.1 *For all a and b with $a \geq b \geq \epsilon$ and $a + b \leq n - k$, the matrix*

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_{b+1} \\ S_2 & S_3 & \cdots & S_{b+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_a & S_{a+1} & \cdots & S_{a+b} \end{pmatrix}$$

has rank ϵ.

Proof The lemma holds vacuously if $\epsilon > t$, so assume $\epsilon \leq t$. Using that $S_i = e(\alpha^i) = \sum_{j=0}^{n-1} \alpha^{ij} e_j$, we can expand the matrix to a matrix product:

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_{b+1} \\ S_2 & S_3 & \cdots & S_{b+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_a & S_{a+1} & \cdots & S_{a+b} \end{pmatrix} =$$

$$\begin{pmatrix} 1 & \alpha^1 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \cdots & (\alpha^2)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^a & \cdots & (\alpha^a)^{n-1} \end{pmatrix} \begin{pmatrix} e_0 & e_0 & \cdots & e_0 \\ e_1 & \alpha e_1 & \cdots & \alpha^b e_1 \\ \vdots & \vdots & \ddots & \vdots \\ e_{n-1} & \alpha^{n-1} e_{n-1} & \cdots & (\alpha^b)^{n-1} e_{n-1} \end{pmatrix} =$$

$$\begin{pmatrix} 1 & \alpha^1 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \cdots & (\alpha^2)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^a & \cdots & (\alpha^a)^{n-1} \end{pmatrix} \begin{pmatrix} e_0 & \cdots & 0 & 0 \\ \vdots & e_1 & \cdots & 0 \\ 0 & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha & \cdots & \alpha^b \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-1} & \cdots & (\alpha^b)^{n-1} \end{pmatrix} =$$

$$\underbrace{\begin{pmatrix} \alpha^{\ell_1} & \alpha^{\ell_2} & \cdots & \alpha^{\ell_\epsilon} \\ (\alpha^2)^{\ell_1} & (\alpha^2)^{\ell_2} & \cdots & (\alpha^2)^{\ell_\epsilon} \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha^a)^{\ell_1} & (\alpha^a)^{\ell_2} & \cdots & (\alpha^a)^{\ell_\epsilon} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} e_{\ell_1} & \cdots & 0 & 0 \\ \vdots & e_{\ell_2} & \cdots & 0 \\ 0 & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e_{\ell_\epsilon} \end{pmatrix}}_{B} \underbrace{\begin{pmatrix} 1 & \alpha^{\ell_1} & \cdots & (\alpha^{\ell_1})^b \\ 1 & \alpha^{\ell_2} & \cdots & (\alpha^{\ell_2})^b \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{\ell_\epsilon} & \cdots & (\alpha^{\ell_\epsilon})^b \end{pmatrix}}_{C}$$

In the last step, $\ell_1, \ldots, \ell_\epsilon$ are the error positions, and the step holds because the remaining e_i are all zero. The matrix C is a partial Vandermonde matrix and as all $\alpha^i, i = 0, \ldots, n-1$ are different, and as $b \geq \epsilon$, it must have full row rank ϵ; seen as a linear map, it is therefore surjective with an image of rank ϵ. The matrix B has only non-zero elements on the diagonal and therefore has full rank, so it is an isomorphism map, and then BC is also a surjective map of rank ϵ. The last matrix A is also a partial Vandermonde, so, as $a \geq \epsilon$, it has full column rank ϵ, and is therefore an injective map with an image of dimension ϵ. Thus, the map $A(BC)$ must also have an image of dimension ϵ and is therefore of rank ϵ. □

Now we can prove that we can find a polynomial "close enough" to $\Lambda(x)$, simply by solving a homogeneous system of linear equations on the syndromes, invariant of ϵ:

Theorem 2.3.2 *If $\epsilon \le t$, then for any solution $(v_0, \dots, v_t)$ to $(u_0, \dots, u_t)$ in the matrix equation*

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_{t+1} \\ S_2 & S_3 & \cdots & S_{t+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_t & S_{t+1} & \cdots & S_{2t} \end{pmatrix} \begin{pmatrix} u_t \\ u_{t-1} \\ \vdots \\ u_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{2.3.1}$$

the induced polynomial $v(x) = \sum_{i=0}^{t} v_i x^i$ satisfies $\Lambda(x) \mid v(x)$

Proof Let $\Lambda = (\Lambda_0, \dots, \Lambda_\epsilon)$ be the vector of the coefficients of $\Lambda(x)$. We can write (2.2.1) in matrix form, and see that Λ, and therefore also $k\Lambda$ for all $k \in \mathbb{F}_q$, solve for w the equation

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_{\epsilon+1} \\ S_2 & S_3 & \cdots & S_{\epsilon+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_{2t-\epsilon} & S_{2t-\epsilon+1} & \cdots & S_{2t} \end{pmatrix} \begin{pmatrix} w_\epsilon \\ w_{\epsilon-1} \\ \vdots \\ w_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

from which we can select sub-matrices, yielding the following set of equations:

$$\begin{pmatrix} S_{i+1} & S_{i+2} & \cdots & S_{i+\epsilon+1} \\ S_{i+2} & S_{i+3} & \cdots & S_{i+\epsilon+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_{i+t} & S_{i+t+1} & \cdots & S_{i+t+\epsilon} \end{pmatrix} \begin{pmatrix} w_\epsilon \\ w_{\epsilon-1} \\ \vdots \\ w_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad i = 0, \dots, t-\epsilon$$

But for all the i, the S matrix in the above is a contiguous part of the columns of the matrix in (2.3.1). Therefore, $(u_0, \dots, u_t) = (0, \dots, k\Lambda_0, \dots, k\Lambda_\epsilon, \dots, 0)$ is a solution to (2.3.1) for any number of leading zeroes $i = 0, \dots, t-\epsilon$.

The equation (2.3.1) is a homogeneous system of t linear equations in $t+1$ variables. From Lemma 2.3.1, the S-matrix has rank ϵ. Therefore, the solution set has exactly $t-\epsilon+1$ independent parameters. But all the $(0, \dots, k\Lambda_0, \dots, k\Lambda_\epsilon, \dots, 0)$ are independent for each i, and i can attain precisely $t-\epsilon+1$ values; therefore, these parameters account for all the solutions to the system.

A solution v therefore has the form

$$(v_0, \dots, v_t) = \begin{array}{ll} & k_0(\Lambda_0, \dots, \Lambda_\epsilon, \quad \dots \quad, 0) \\ + & k_1(0, \Lambda_0, \dots, \Lambda_\epsilon, \dots, 0) \\ \vdots & \vdots \\ + & k_{t-\epsilon}(0, \quad \dots \quad, \Lambda_0, \dots, \Lambda_\epsilon) \end{array}$$

with all $k_i \in \mathbb{F}_q$. This means that the induced polynomial has the form

$$\begin{aligned} v(x) &= k_0\Lambda(x) + xk_1\Lambda(x) + \dots + x^{t-\epsilon}k_{t-\epsilon}\Lambda(x) \\ &= \Lambda(x)(k_0 + xk_1 + \dots + x^{t-\epsilon}k_{t-\epsilon}), \end{aligned}$$

proving the theorem. □

The main use of $\Lambda(x)$ is that each of its zeroes reveal an error position, so if we obtain a relatively low-degree polynomial $p(x) = q(x)\Lambda(x)$, most of the zeroes of $p(x)$ will reveal error positions and only a few will be misleading. If $p(x)$ has degree at most t, it turns out that Forney's algorithm can simultaneously sort out all these misleading zeroes and find the error values for the true zeroes.

We then already have a minimum-distance decoder, as any matrix-equation solving algorithm – e.g. Gaussian Elimination – can be used for the equation of Theorem 2.3.2 to obtain a $p(x) = q(x)\Lambda(x)$ with degree at most t.

However, it turns out that the so-called Hankel form of the S-matrix in (2.3.1) allows for a much faster method for solving the equation. The method is basically the Euclidean algorithm done on polynomials, but with a small twist at the ending. We will first give a brief description of this algorithm, so we are well-enough equipped to show the decoder afterwards.

2.3.1 Euclid lends a hand

For understanding the Sugiyama algorithm, we briefly have to introduce the Extended Euclidean algorithm for polynomials (EA). The reader who is familiar with this algorithm can skip to the end of this section; there we present two small results, which exceeds in detail what is usually necessary to know when applying the EA algorithm.

The algorithm is run on input polynomials $a(x), b(x)$ with $\deg(a(x)) \geq \deg(b(x))$ and it returns three polynomials $f(x), g(x), r(x)$ which satisfy

$$f(x)a(x) + g(x)b(x) = r(x) = \gcd(a(x), b(x))$$

Algorithm 2.3.3 (Extended Euclidean for polynomials)
Initialise with

$$\begin{aligned} r^{(-1)}(x) &= a(x) \qquad & f^{(-1)}(x) &= -1 \qquad & g^{(-1)}(x) &= 0 \\ r^{(0)}(x) &= b(x) & f^{(0)}(x) &= 0 & g^{(0)}(x) &= 1 \end{aligned}$$

Continue in a loop for $i = 1, \ldots$. In each iteration, perform first polynomial division with remainder and obtain the $r^{(i)}(x)$ which satisfies

$$r^{(i-2)}(x) = q^{(i)}(x)r^{(i-1)}(x) + r^{(i)}(x), \qquad \deg(r^{(i)}(x)) < \deg(r^{(i-1)})$$

If $r^{(i)}(x) \neq 0$ then update f and g as

$$\begin{aligned} f^{(i)}(x) &= f^{(i-2)}(x) - q^{(i)}(x)f^{(i-1)}(x) \\ g^{(i)}(x) &= g^{(i-2)}(x) - q^{(i)}(x)g^{(i-1)}(x) \end{aligned}$$

If instead $r^{(i)}(x) = 0$, break and return $\big(f^{(i-1)}(x), g^{(i-1)}(x), r^{(i-1)}(x)\big)$.

■

The degree of the $r^{(i)}(x)$ is ever decreasing, so for some iteration, the algorithm will break. That the algorithm produces what is expected follows from the following proposition, which also ensures similar properties for the intermediate results. We will not prove these but the proofs are well-known and not hard; see for example [14].

Proposition 2.3.4 *For all $i > -1$*

1. $\gcd(r^{(i+1)}(x), r^{(i)}(x)) = \gcd(r^{(i)}(x), r^{(i-1)}(x))$
2. $f^{(i)}(x)a(x) + g^{(i)}(x)b(x) = r^{(i)}(x)$

Furthermore, for proving the Sugiyama algorithm, we need a third invariant

Lemma 2.3.5 *For all $i > -1$,* $\deg(g^{(i)}(x)) + \deg(r^{(i-1)}(x)) = \deg(a(x)))$

2.3.2 Using the EA for finding $\Lambda(x)$ (or close enough)

Returning to coding theory, we are now capable of showing a faster way to solve the matrix equation in Theorem 2.3.2. Define $\grave{S}(x)$ as the polynomial induced by the first $2t$ syndromes, that is $\grave{S}(x) = \sum_{i=0}^{2t} x^i S_{i+1}$; these are exactly the syndromes present in the said matrix equation. Moving Theorem 2.3.2 back into polynomials, we can arrive at the following result:

Corollary 2.3.6 *If $\epsilon \leq t$, then any polynomial $p(x)$ of degree at most t satisfying*

$$\deg\left(p(x)\grave{S}(x) \bmod x^{2t}\right) < t$$

has $\Lambda(x) \mid p(x)$

Proof Assume $p(x)$ is of degree $s \leq t$ and has $\deg\left(p(x)\grave{S}(x) \bmod x^{2t}\right) < t$. This means that the coefficients to terms $t, \ldots, 2t-1$ of the product $p(x)\grave{S}(x)$ are all zero. Letting $p(x) = \sum_{i=0}^{s} p_i x^i$, this gives the following set of equations:

$$\sum_{i=0}^{s} p_i S_{j-i} = 0, \quad j = t+1, \ldots, 2t$$

But then the sequence $(p_0, \ldots, p_s, 0, \ldots, 0)$ with $t - s$ zeroes is a solution to the equation of Theorem 2.3.2. From that theorem we then have $\Lambda(x) \mid p(x)$. □

We are now very close; the decoder comes from simply realising that we can satisfy the requirements of the above corollary by stopping the EA short:

Theorem 2.3.7 *Assume $\epsilon \leq t$. Run the Extended Euclidean algorithm on input $(a(x), b(x)) = (x^{2t}, \grave{S})$, and let i be the first iteration with $\deg(r^{(i)}(x)) < t$. Then $\Lambda(x) \mid g^{(i)}(x)$.*

Proof We know $\deg(r^{(i-1)}(x)) \geq t$, so from Lemma 2.3.5 on the preceding page, we have

$$\deg(g^{(i)}(x)) = \deg(x^{2t}) - \deg(r^{(i-1)}(x)) \leq 2t - t = t$$

Furthermore, from Proposition 2.3.4, we have

$$\begin{aligned} f^{(i)}(x)x^{2t} + g^{(i)}(x)\grave{S}(x) = r^{(i)}(x) & \qquad \Longrightarrow \\ g^{(i)}(x)\grave{S}(x) \equiv r^{(i)}(x) \mod x^{2t} & \qquad \Longrightarrow \\ \deg(g^{(i)}(x)\grave{S}(x) \bmod x^{2t}) < t & \end{aligned}$$

and the result follows from Corollary 2.3.6. □

We then arrive at this thesis' first decoding algorithm:

Algorithm 2.3.8 (Sugiyama decoding)

1. Run the Extended Euclidean algorithm on input $(a(x), b(x) = (x^{2t}, \grave{S})$, and stop at the first iteration i with $\deg(r^{(i)}(x)) < t$.
2. Find the roots of $g^{(i)}(x)$. Use Forney's algorithm to find the error values while removing the roots of $g^{(i)}(x)$ which are not roots of $\Lambda(x)$.
3. If the result is a valid codeword within distance t of r, return the corresponding information word. Otherwise, declare a decoding failure.

■

2.4 Linear recursions and the BMA

The ultimate goal of this chapter is to give the reader a full understanding of the Berlekamp-Massey decoding algorithm, and we will now return to that. Though this was not the historical order of development, we will present the decoder from the view of linear recursions, resembling the one first given by Massey in [17]. This section will end with the Berlekamp-Massey algorithm for solving linear recursions, which is the essence of the decoding algorithm.

In the literature, both the linear-recursion solver and the decoder carry the name "the Berlekamp-Massey algorithm", and they are also often both shortened "BMA". In this thesis, we will use the convention of calling the linear-recursion solver "the

Berlekamp-Massey algorithm" or simply "BMA", while the decoding algorithm is "the Berlekamp-Massey decoder".

We will need quite a few general results on linear recursions, and this section contains no references to the decoding problem. For the reader familiar with linear recursions it would still be a good idea to skim through this section for the notational conventions introduced; they will be used heavily when deriving Wu's algorithm in Chapter 4. This goes especially for the notation of the BMA.

Linear recursions are a mathematical concept found in many places, but the presentation given here, and the main problem solved, is especially inspired by an equivalent problem in electronics: that of finding the shortest linear-feedback shift register (LFSR) producing some sequence. Indeed, a large part of Massey's paper presenting this approach to the Berlekamp-Massey decoder is concerned with details in electronics. Here, though, we will let it be a curiosity, mentioning it only for the reader interested in an intuitive and concrete demonstration of linear recursions; see for example [4].

A sequence $V = V_1, V_2, \ldots$ is called *linearly recursive* if there exists a length L and constant coefficients $A_1, \ldots, A_L$ such that the following *linear recursion* holds:

$$V_i = -\sum_{j=1}^{L} A_j V_{i-j}, \quad i = L+1,\ L+2, \ldots, \tag{2.4.1}$$

The V_i and A_i are all elements of some field $\mathbb{F}$. When speaking of V, as the equation is then completely given by the A_i, we refer to them simply as "the recursion", and we say that they produce V given the terms $V_1, \ldots, V_L$.

If L is the smallest number for which a linear recursion of length L over a sequence V exists, then L is said to be the *linear complexity* of V and is denoted $\mathcal{L}(V)$. If V is an all-zero sequence, then we define $\mathcal{L}(V) = 0$, and if V is not linearly recursive, we define $\mathcal{L}(V) = \infty$. For a finite sequence V of length m, it is trivial to find a linear recursion of length $m-1$, so we have $\mathcal{L}(V) < m$; thus, all not linearly-recursive sequences must be infinite.

These definitions naturally gives rise to an interesting problem:

Problem 2.4.1 *Given a sequence V, find the linear complexity of V, and if this has a finite value L, find a recursion of length L producing V.*

The BMA solves the above problem for all finite sequences, and as these are all linearly recursive, it always finds a recursion. Note that the recursion is generally not unique and the BMA just finds one.

Before we can derive the algorithm, we need three results on linear recursions. First a basic results about linear complexity, which will later enable us to prove the important Berlekamp-Massey Theorem that directly leads to the algorithm.

Proposition 2.4.2 *Let V be a sequence of length m. If the recursions A of length L and A' of length L' both produce V, and if $m \geq L + L'$, then they will always produce the same sequence.*

Proof We prove that, given the premises, then A and A' will next produce the same element. As the premises still hold afterwards, it follows directly that they will then always produce the same elements.

Let $V = V_1, \ldots, V_m$. Let also $A = A_1, \ldots, A_L$ and lastly let $A' = A'_1, \ldots, A'_{L'}$. Writing out the expressions for the sequences' $m + 1$'th production, we have to prove

$$-\sum_{j=1}^{L} A_j V_{m+1-j} = -\sum_{k=1}^{L'} A'_k V_{m+1-k} \tag{2.4.2}$$

The left-hand side depends on the elements $V_{m+1-L}, \ldots, V_m$ from V, and as $m \geq L + L' \iff L' + 1 \leq m + 1 - L$, these elements are all produced by A' according to the linear recursion expression of (2.4.1):

$$V_i = -\sum_{k=1}^{L'} A'_k V_{i-k}, \quad i = m+1-L, \ldots, m$$

We can therefore expand V_{m+1-j} in the left-hand side of (2.4.2):

$$-\sum_{j=1}^{L} A_j V_{m+1-j} = -\sum_{j=1}^{L} \left(A_j \left(-\sum_{k=1}^{L'} A'_k V_{m+1-j-k} \right) \right)$$

Now also $L + 1 \leq m + 1 - L'$, so similarly to before, the elements $V_{m+1-L'}, \ldots, V_m$ are produced by A according to its linear recursion expression. Therefore, we can rewrite the expression and contract the resulting inner sum to V_{m+1-k}:

$$\begin{aligned} -\sum_{j=1}^{L} \left(A_j \left(-\sum_{k=1}^{L'} A'_k V_{m+1-j-k} \right) \right) &= -\sum_{k=1}^{L'} \left(A'_k \left(-\sum_{j=1}^{L} A_j V_{m+1-k-j} \right) \right) \\ &= -\sum_{k=1}^{L'} A'_k V_{m+1-k} \end{aligned}$$

□

Before we continue, we introduce a convenient notation for linear recursions. With a recursion's connection coefficients $A = A_1, \ldots, A_L$, we associate a *connection polynomial* $A(x) = 1 + A_1 x + \ldots + A_L x^L$. It turns out that many properties of linear recursions are most naturally explored and expressed using the connection polynomial. Note that A_L is possibly zero[1], which would make the degree of $A(x)$ less than

[1]At a glance, it might seem that we could require $A_L \neq 0$, as this coefficient would otherwise not contribute to the linear recursion in (2.4.1). However, in this case, the length L can be seen as "from which element in V and forwards the recursion holds", which might well be larger than the number of non-zero coefficients.

L, so to identify the recursion with the connection polynomial, we must also supply L; when identifying recursions like this, we often use the notation $(A(x), L)$.

The constant coefficient of 1 makes perfect sense when rewriting (2.4.1):

$$V_i = -\sum_{j=1}^{L} A_j V_{i-j} \iff V_i + \sum_{j=1}^{L} A_j V_{i-j} = 0 \iff \sum_{j=0}^{L} A_j V_{i-j} = 0, \qquad (2.4.3)$$

if we define $A_0 = 1$.

This perspective also enables us to understand how certain arithmetic on the polynomial forms influence the recursion. Assume that $A(x)$ and $B(x)$ as recursions of length L and $L-1$ respectively both produce some sequence V. The addition $A(x) + B(x)$ does not have 1 as constant coefficient, and so does not directly make sense as a linear recursion (in our definition). However $A(x) + xB(x)$ *does* make sense, and the productions over $V_1, \ldots, V_L$ have the form

$$\begin{aligned} V'_i &= -\sum_{j=1}^{L}(A_j + B_{j-1})V_{i-j}, \qquad\qquad i = L+1, L+2, \ldots \\ &= -\sum_{j=1}^{L} A_j V_{i-j} - \sum_{k=0}^{L-1} B_k V_{i-1-k}, \end{aligned}$$

where A_j and B_j are the coefficients of the j'th term of $A(x)$ respectively $B(x)$. Remembering (2.4.3) and that $B(x)$ produces V, the second sum must be zero, so

$$V'_i = -\sum_{j=1}^{L} A_j V_{i-j}$$

But as $A(x)$ produces V, we must have $V'_i = V_i$, which means that $(A(x)+xB(x), L)$ also produces V. This demonstrates the linearity of the recursions.

We are now ready for another more complicated and specific lemma, which provides the main update mechanism of the iterations of the BMA:

Lemma 2.4.3 *With $V_1, \ldots, V_r, \ldots, V_m$ a sequence, let $A = A_1, \ldots, A_{L_A}$ be a recursion producing $V_1, \ldots, V_{m-1}$ but not V_m next, and similarly $B = B_1, \ldots, B_{L_B}$ be a recursion producing $V_1, \ldots, V_{r-1}$, but not V_r next. Furthermore, let the (non-zero) discrepancies with regards to the next productions δ_A and δ_B be given by*

$$\delta_A = V_m - \Big(-\sum_{j=1}^{L_A} A_j V_{m-j} \Big) \qquad\qquad \delta_B = V_r - \Big(-\sum_{k=1}^{L_B} B_k V_{r-k} \Big)$$

Then a recursion of length $\max(L_A,\ L_B + m - r)$ *induced by the polynomial*

$$C(x) = A(x) - \frac{\delta_A}{\delta_B} x^{m-r} B(x)$$

produces $V_1, \ldots, V_m$.

Proof We need to prove that the recursion corresponding to $C(x)$ and of length $L = \max(L_A,\ L_B + m - r)$ produces $V_{L+1}, \dots, V_m$. Of course, this is vacuously true if not $L < m$, but as the linear complexity of a finite sequence is less than the length of the sequence, both $L_A < m$ and $L_B < r$, which implies the inequality.

Following the above discussion, we see that $C(x)$ does actually correspond to a recursion, as the constant term is 1, which is the only requirement. Slightly generalising parts of that discussion, we can define $B_0 = 1$ and also easily find the expression for the productions over $V_1, \dots, V_L$:

$$V_i' = -\left(\sum_{j=1}^{L_A} A_j V_{i-j} - \frac{\delta_A}{\delta_B} \sum_{k=0}^{L_B} B_k V_{i-(m-r)-k} \right), \quad i = L+1, L+2, \dots$$

We are sure that the indices on the V terms in the second sum are positive, as

$$i > L \geq L_B + m - r \geq m - r + k$$

We wish to prove that $V_i' = V_i$ for $i = L+1, \dots, m$. Now, for all but the last of these productions, we can set $h = i - (m - r)$ in the second sum, and get

$$\sum_{k=0}^{L_B} B_k V_{i-(m-r)-k} = \sum_{k=0}^{L_B} B_k V_{h-k}\ , \qquad h = L - (m-r) + 1, \dots, r-1$$

This has exactly the form of the left-hand side of (2.4.3), so as $B(x)$ produces V_h for all these h, the expression must be zero for all of them. This leaves

$$V_i' = -\sum_{j=1}^{L_A} A_j V_{i-j} = V_i, \quad i = L+1, L+2, \dots, m-1$$

Left is then only to prove $V_m' = V_m$. But here we have

$$\begin{aligned} V_m' &= -\left(\sum_{j=1}^{L_A} A_j V_{m-j} - \frac{\delta_A}{\delta_B} \sum_{k=0}^{L_B} B_k V_{m-(m-r)-k} \right) \\ &= -\sum_{j=1}^{L_A} A_j V_{m-j} + \frac{\delta_A}{\delta_B} \sum_{k=0}^{L_B} B_k V_{r-k} \\ &= V_m - \delta_A + \frac{\delta_A}{\delta_B} \delta_B \\ &= V_m \end{aligned}$$

□

We will now prove a surprising and elegant theorem, whose word and proof directly gives the BMA, which we present afterwards:

Theorem 2.4.4 (Berlekamp-Massey) *Let $V = V_1, \dots, V_m$ be a sequence whose first $m-1$ elements have $\mathcal{L}(V_1, \dots, V_{m-1}) = L$. If a recursion of length L produces $V_1, \dots, V_{m-1}$ but does not next produce V_m, then $\mathcal{L}(V) = \max(L,\ m - L)$.*

Proof Let $A = A_1, \dots, A_L$ be a recursion producing $V_1, \dots, V_{m-1}$ but not next V_m.

We first prove that $\mathcal{L}(V) \geq \max(L,\ m - L)$. Let A' be a recursion of length L' producing $V_1, \dots, V_m$. We easily have that $L' \geq L$ as A' also produces $V_1, \dots, V_{m-1}$ which has linear complexity L. Furthermore, as A and A' agree on the first $m-1$ productions but not on the m'th, Proposition 2.4.2 implies that $m \leq L' + L \iff L' \geq m - L$.

Now we can prove $\mathcal{L}(V) = \max(L,\ m - L)$. Define a *complexity-increasing index* as an index i where $\mathcal{L}(V_1, \dots, V_{i-1}) < \mathcal{L}(V_1, \dots, V_i)$. We proceed by induction on the number of complexity-increasing indices before m.

The base case is with zero such indices. As the empty sequence as well as any zero-sequence has linear complexity 0, while a sequence with any non-zero element has positive linear complexity, sequences with no complexity-increasing indices before m must have the form $V = 0, \dots, 0, x$, with possibly 0 zeroes. Therefore, A must be the empty recursion, and as it does not produce x, we also have $x \neq 0$. But then x cannot be produced by a linear combination of the $m-1$ zeroes, so V necessarily has linear complexity $m = \max(0,\ m - 0)$, which is what we sought.

Now for the inductive step. The induction hypothesis is that for all complexity-increasing indices $i < m$, then $\mathcal{L}(V_1, \dots, V_i) = i - \mathcal{L}(V_1, \dots, V_{i-1})$, and we must show that this implies $\mathcal{L}(V) = \max(L,\ m - L)$; notice that this will also maintain our induction hypothesis.

Let r be the highest complexity-increasing index less than m. Then there exists a recursion B of minimum length L_B producing $V_1, \dots, V_{r-1}$ but not V_r next. Using A and B in Lemma 2.4.3, there exists a linear recursion producing V and length $L_C = \max(L,\ L_B + m - r)$. But by the induction hypothesis,

$$\begin{aligned}
\mathcal{L}(V_1, \dots, V_{m-1}) = \mathcal{L}(V_1, \dots, V_r) = r - \mathcal{L}(V_1, \dots, V_{r-1}) \qquad & \iff \\
L = r - L_B \qquad & \implies \\
L_B = r - L \qquad &
\end{aligned}$$

so $L_C = \max(L,\ m - L) \implies \mathcal{L}(V) \leq \max(L,\ m - L)$. By the previously proved lower bound for $\mathcal{L}(V)$, we get the desired result. □

We can now derive the BMA. To describe it informally: it proceeds iteratively over the length of V and maintains a current recursion $A^{(i)}$ which produces $V_1, \dots, V_i$ and has minimum length. If $A^{(i)}$ produces V_{i+1} next, then $A^{(i)}$ must also be a minimum length recursion producing $V_1, \dots, V_{i+1}$. If it does not, however, we do as in the proof: retrieve the last complexity-increasing index r and a recursion $A^{(r-1)}$ producing $V_1, \dots, V_{r-1}$ – this was computed at an earlier iteration – and compute $A^{(i+1)}$ from $A^{(i)}$ and $A^{(r-1)}$ as according to Lemma 2.4.3.

This can be described in a very succinct algorithm:

Algorithm 2.4.5 (Berlekamp-Massey)
Initialise with

$$A^{(0)}(x) = 1 \qquad L^{(0)} = 0 \qquad B^{(0)}(x) = 0$$

Do the following for each $i = 1, \dots, m$. First compute a discrepancy as

$$\delta^{(i)} = \sum_{j=0}^{L^{(i-1)}} A_j^{(i-1)} V_{i-j}$$

Then, if $\delta^{(i)} = 0$, compute new values as

$$\begin{aligned} A^{(i)}(x) &= A^{(i-1)}(x) \\ L^{(i)} &= L^{(i-1)} \\ B^{(i)}(x) &= xB^{(i-1)}(x) \end{aligned}$$

Otherwise, compute new values as[2]

$$\begin{aligned} A^{(i)}(x) &= A^{(i-1)}(x) - \delta^{(i)} x B^{(i-1)}(x) \\ L^{(i)} &= \max(L^{(i-1)},\ i - L^{(i-1)}) \\ B^{(i)}(x) &= xB^{(i-1)}(x)[L^{(i)} = L^{(i-1)}] + \frac{1}{\delta^{(i)}} A^{(i)}(x)[L^{(i)} \neq L^{(i-1)}] \end{aligned}$$

Then $(A^{(m)}(x), L^{(m)})$ is a minimum-length recursion producing V.
■

The algorithm remembers the recursion of the last complexity-increasing index in $B^{(i)}(x)$, but multiplied with the $\frac{1}{\delta^{(i)}}$ factor for when it is to be used according to Lemma 2.4.3. It also iteratively adds the x^{m-r} factor by multiplying by x in each iteration. This way, all information needed for invoking Lemma 2.4.3 is accumulated in $B^{(i)}(x)$.

We will not formally prove that the algorithm is correct; with the results already given, the proof is not hard: it merely establishes that $B^{(i)}(x)$ has the form informally described above, and from then invokes Lemma 2.4.3 and Theorem 2.4.4.

Most often, only the final $(A^{(m)}(x), L^{(m)})$ is of interest, but sometimes also the $B^{(m)}(x)$ or even the intermediate results provides information. This is e.g. the case for Wu's decoding algorithm, presented in Chapter 4.

As a last remark, the following corollary describes how the calculation of $A^{(i)}(x)$ and $B^{(i)}(x)$ can be written in a matrix form, which sometimes makes the algorithm easier to argue about:

[2]We use the Iverson bracket, where $[true]$ is 1 and $[false]$ is a "strong" 0. See Knuth [15].

Corollary 2.4.6 *Consider the i'th iteration of the BMA on some input, $i > 0$. Define first the following binary variable*

$$\flat^{(i)} = [\delta^{(i)} \neq 0 \ \wedge \ L \neq L^{(i-1)}]$$

Then

$$\begin{pmatrix} A^{(i)}(x) \\ B^{(i)}(x) \end{pmatrix} = \begin{pmatrix} 1 & -\delta^{(i)}x \\ \flat^{(i)}(\delta^{(i)})^{-1} & (1-\flat^{(i)})x \end{pmatrix} \begin{pmatrix} A^{(i-1)}(x) \\ B^{(i-1)}(x) \end{pmatrix},$$

Proof By calculating the different cases of the progression of the algorithm. □

2.5 Decoding with Berlekamp and Massey

Well equipped with tools for tackling linear recursions, we will now set up the decoding problem as a linear recursion problem, and the decoder works simply by solving this linear recursion. As we will see, from the derivation emerges the restriction that the Berlekamp-Massey decoding algorithm only works as long as $\epsilon \leq t$.

The setup is quite simple. Remember from (2.2.1) on page 15 that as long as $\epsilon < n-k$, the error-locator $\Lambda(x)$ satisfies the following set of equations:

$$\sum_{j=0}^{\epsilon} \Lambda_j S_{i-j} = 0, \quad i = \epsilon+1, \ldots, n-k$$

As $\Lambda_0 = 1$ we can instead write

$$S_i = -\sum_{j=1}^{\epsilon} \Lambda_j S_{i-j}, \quad i = \epsilon+1, \ldots, n-k$$

After Section 2.4, we immediately recognise this as a linear recursion producing S given $(\Lambda(x), \epsilon)$. That is not all; we can now prove the stronger and central result of the Berlekamp-Massey decoder:

Theorem 2.5.1 *If $\epsilon \leq t$, then $(\Lambda(x), \epsilon)$ is the unique shortest linear recursion producing S.*

Proof From the discussion above, we already know that $(\Lambda(x), \epsilon)$ produces S whenever $\epsilon \leq t < n-k$. To prove that it is the shortest, assume oppositely that a recursion $u = u_1, \ldots, u_m$ with $m < \epsilon$ produces S. If we define $u_0 = 1$, moving all terms of the linear recursion equation to the left-hand side, as in (2.4.3) on page 23, reveals that u satisfies the following matrix equation:

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_{m+1} \\ S_2 & S_3 & \cdots & S_{m+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n-k-m} & S_{n-k-m+1} & \cdots & S_{n-k} \end{pmatrix} \begin{pmatrix} u_m \\ u_{m-1} \\ \vdots \\ u_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This means that $(v_0, \dots, v_\epsilon) = (0, \dots, 0, u_0, \dots, u_m)$ must be a solution to this wider but shorter matrix equation

$$\begin{pmatrix} S_1 & S_2 & \cdots & S_{m+1} & \cdots & S_{\epsilon+1} \\ S_2 & S_3 & \cdots & S_{m+2} & \cdots & S_{\epsilon+2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ S_\epsilon & S_{\epsilon+1} & \cdots & S_{\epsilon+m-1} & \cdots & S_{2\epsilon} \end{pmatrix} \begin{pmatrix} v_\epsilon \\ v_{\epsilon-1} \\ \vdots \\ v_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

Note that all elements of S used in the matrix are defined because $2\epsilon \leq 2t \leq n-k$. Likewise, the matrix is shorter as $n-k-m > n-k-\epsilon \geq \epsilon$.

From (2.2.1), we already know that this system of equations has among its solutions Λ and therefore also $k\Lambda$ for all $k \in \mathbb{F}_q$. But it is a homogeneous system of ϵ linear equations over $\epsilon + 1$ unknowns, and Lemma 2.3.1 on page 16 asserts that the S-matrix in the equation has full rank ϵ. This means that the solution set has only one free parameter, so $k\Lambda$ must account for *all* solutions, rendering impossible the solution $(0, \dots, 0, u_0, \dots, u_m)$.

Lastly, a solution $v_0, \dots, v_\epsilon$ to the matrix equation is only a linear recursion producing S if $v_0 = 1$. Therefore, of the $k\Lambda$ solutions, only Λ satisfies the requirements of the theorem. □

The theorem implies, that if $\epsilon \leq t$ and we run the BMA on S, it will return Λ. The minimum-distance decoder then readily emerges:

Algorithm 2.5.2 (Berlekamp-Massey decoding)

1. Run the BMA on S, which will return a recursion $(A(x), L)$
2. If either $L \neq \deg(A(x))$ or $L > t$ or $A(x)$ does not have exactly L distinct non-zero roots, it means $A(x) \neq \Lambda(x)$. This implies $\epsilon > t$, so declare a decoding failure.
3. Otherwise, we guess $A(x) = \Lambda(x)$ and the roots are the error positions; find these, and use Forney's algorithm to find the error values.
4. If the result is a valid codeword, return the corresponding information word. Otherwise, declare a decoding failure.

■

This concludes the presentation of the Berlekamp-Massey decoder. We will see it again when deriving Wu's algorithm in Chapter 4; here we will further investigate the BMA and see that when run on S, it provides us with essential information even when $\epsilon > t$.

Chapter 3

The very first list-decoding algorithms

In 1997 – almost 40 years after the invention of Reed-Solomon codes – there were still no polynomial-time algorithms found which could decode significantly beyond half the minimum distance. Then Sudan published a ground-breaking paper [22], where the decoding problem was modelled as a polynomial interpolation problem, leading to a proof-of-concept algorithm which could improve on minimum-distance decoding provided that the rate $\frac{k}{n}$ was low. The algorithm was much slower than the known minimum-distance decoders, but the barrier had been broken and the fresh approach incited a renewed interest in Reed-Solomon codes. Only two years later, it was Sudan himself and his graduate student Guruswami who generalised the algorithm to improve the decoding bound for all rates. This is the famous Guruswami-Sudan algorithm.

As described it in Section 1.1, list-decoding algorithms are so named because when decoding beyond half the minimum distance, several codewords might be equally close to the received word, and the decoder must return the list of these. Such a decoding algorithm must be controlled by a parameter τ – the error-correction capacity – which sets the maximum number of errors that we wish to be able to correct; a polynomial time decoder cannot have this too high as it would result in a super-polynomial number (in n) of codewords on the list. As we also mentioned in Section 1.2.2, $n - k$ is a definite upper bound for the value of τ, but whether the actual bound is tighter for Reed-Solomon codes is unknown.

For the Guruswami-Sudan decoding algorithm, τ is bounded by the so-called Johnson bound: $\tau < n - \sqrt{n(k-1)}$. Most improvements on the algorithm have been for speed efficiency, and have not touched this bound. However, recently in [18], the bound was crossed by a little at a massive price of speed, which indicates that the bound is not sacrosanct.

Both the Sudan and the Guruswami-Sudan have another parameter: the list size l. It emerges from the algorithm as a parameter for the complexity of the decoding attempt, but its name comes from its most striking property: it is an upper bound on the number of possible information words which can be returned by running the algorithm. Therefore, it is of course strongly related to τ and the two cannot be set freely.

In this chapter we will present and prove first the Sudan algorithm and then the generalised Guruswami-Sudan. The presentation we will give here differs slightly from the original papers and is strongly inspired by that of Justesen and Høholdt [14]. Both algorithms perform decoding by modelling it as a problem of polynomial interpolation. We will therefore begin the chapter with a short description of this problem in Section 3.1. Then in Section 3.2, we will give the Sudan algorithm, followed by the Guruswami-Sudan algorithm in Section 3.3. The algorithms are very related so their presentation are closely mirrored. For both algorithms, we will shortly analyse on the limitations on the choices of their parameters.

3.1 Polynomial interpolation and decoding

At the heart of both the Sudan and the Guruswami-Sudan algorithm is the rephrasing of the decoding problem into one of finding interpolating polynomials. We can state this problem as:

Problem 3.1.1 (Polynomial interpolation) *Given positive integers n, k and τ with $n > \tau$, as well as n distinct points $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$, find all polynomials of degree less than k which go through at least $n - \tau$ of those points.*

Here we overload the meaning of both n, k and τ from coding theory, but as we will shortly see, this causes no confusion.

It is not too difficult to see how we can regard decoding of Reed-Solomon codes as polynomial interpolation. Remember first that the codeword c is the information word polynomial $w(x) = \sum_{i=0}^{n-1} w_i x^i$ evaluated at each of the different powers of α:

$$c = \big((w(\alpha^0), w(\alpha^1), \ldots, w(\alpha^{n-1})\big)$$

The received word r equals c at exactly $n - \epsilon$ of the positions; this means that $w(x)$ as a polynomial goes through $n - \epsilon$ of the points $\big((\alpha^0, r_0), \ldots, (\alpha^{n-1}, r_{n-1})\big)$.

Therefore, given an error-correction capability τ, if we have $\epsilon \leq \tau$, then $w(x)$ will be among all interpolating polynomials which go through at least $n - \tau$ of the points of

$\big((\alpha^0, r_0), \ldots, (\alpha^{n-1}, r_{n-1})\big)$. If we then select those of the interpolating polynomials whose corresponding codeword is closest to r, we get a τ-bounded-distance decoder.

This, of course, relies on being able to find the interpolating polynomials for the choice of n, k and τ – and to be usable, this should be done in polynomial time. That is exactly what the Sudan and the Guruswami-Sudan algorithms achieve for some choices of the parameters.

3.2 Sudan ponders alone

Considering decoding modelled as polynomial interpolation, we have a decoding algorithm as soon as we can solve the interpolation problem in polynomial time. The Sudan decoding algorithm is exactly this, equipped with an ingenious method for solving the interpolation problem for some values of n, k and τ.

We will first give the main theorem, which immediately gives a method for solving some cases of the polynomial interpolation problem. We then specialise this for coding theory and write out the resulting Sudan algorithm in full. The remainder of the section contains an analysis on the emergent restrictions on n, k and τ.

Theorem 3.2.1 *For any instance $(n, k, \tau, [(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})])$ of Problem 3.1.1 in some field $\mathbb{F}$, and for some list size $l \geq 1$, consider a non-zero bivariate polynomial $Q(x, y) = \sum_{j=0}^{l} y^j Q_j(x)$ with coefficients from $\mathbb{F}$, that satisfies*

1. *(x_i, y_i) is a zero of $Q(x, y)$ for $i = 0, \ldots, n-1$*
2. *$\deg(Q_j(x)) \leq n - \tau - 1 - j(k-1)$ for $j = 0, \ldots, l$*

Any polynomial $f(x)$ among the solution of the problem instance satisfies

$$(y - f(x)) \mid Q(x, y)$$

Proof Consider the univariate polynomial $Q(x, f(x))$. It has a zero for all x_i where $f(x_i) = y_i$; that is, at least $n - \tau$ zeroes. We can write $Q(x, f(x)) = \sum_{j=0}^{l} Q_j(x) f(x)^j$, so, as $f(x)$ has degree at most $k-1$, by the second requirement, each term of the sum – and therefore the entire polynomial $Q(x, f(x))$ – has degree at most $n - \tau - 1$. But then $Q(x, f(x))$ has more zeroes than its degree so it must be the zero polynomial.

Seeing $Q(x, y)$ as a univariate polynomial in y with coefficients from $\mathbb{F}[x]$, the polynomial $f(x)$ being a zero of the $Q(x, y)$ so regarded implies the sought result. □

The proof naturally explains the bounds for the sub-polynomials $Q_j(x)$ as the maximal bounds which let the proof hold true.

We immediately specialise this to our setting of decoding with the following corollary:

Corollary 3.2.2 *Regard the decoding problem in the setting of polynomial interpolation by using the instance $(n, k, \tau, [(\alpha^0, r_0), \dots, (\alpha^{n-1}, r_{n-1})])$ of Problem 3.1.1, and for this, let $Q(x, y)$ be a bivariate polynomial satisfying the conditions of Theorem 3.2.1. If $\epsilon \leq \tau$ then $Q(x, y)$ satisfies $(y - w(x)) \mid Q(x, y)$.*

Proof Follows from Theorem 3.2.1 and the discussion of Section 3.1. □

Deferring the details till later, we can already now describe the Sudan algorithm in main points:

Algorithm 3.2.3 (Sudan list decoding)

1. Choose a desired value for the error-correction capacity τ and an accompanying appropriate value for the list size l.
2. Construct a $Q(x, y)$ satisfying the conditions of Corollary 3.2.2.
3. Find all factors of $Q(x, y)$ of the form $(y - f(x))$ where $\deg(f(x)) < k$.
4. Return as equally likely information words all of the $f(x)$ whose corresponding codeword has least distance to r and less than τ. If no such word exists, declare a decoding failure.

■

From the above algorithm, we can immediately justify the name "list size" for l: as there can be at most l factors of the form $(y - f(x))$ in $Q(x, y)$, at most l codewords can be returned on the list.

There are some obvious caveats: we need to define what an "appropriate" value for l is, and we need to find a way to construct the $Q(x, y)$. That will be the topic of the remainder of this section.

We also need a method for doing the factoring in step 3; unfortunately, it is outside the scope of this thesis to discuss. Various methods for doing this fast in polynomial time exists; see e.g. [14].

Let us look closer at how and when we can construct such a $Q(x, y)$ polynomial. The strategy is to see the first requirement of Theorem 3.2.1 as n homogeneous linear conditions on the coefficients of $Q(x, y)$. We can then construct $Q(x, y)$ by solving this system. There are definitely non-zero solutions if there are more variables than equations; the number of variables is the number of coefficients we are allowed, which, by the second requirement of Theorem 3.2.1, is at most

$$\sum_{j=0}^{l} \Big(n - \tau - 1 - j(k-1) + 1 \Big) = (l+1)\big(n - \tau - \tfrac{1}{2}(k-1)l\big)$$

As the number of equations is n, we get the requirement

$$\begin{aligned} &n < (l+1)\big(n - \tau - \tfrac{1}{2}(k-1)l\big) && \iff \\ &\tau < n\frac{l}{l+1} - \tfrac{1}{2}l(k-1) && (3.2.1) \end{aligned}$$

The only other requirement we have, is that the system exists; that is, the number of coefficients is not zero. This means that not all of the degree bounds should be negative; the most liberal restriction we can set to guarantee this is for $j = 0$, for which we get the inequality $n > \tau$; but this is already required by the polynomial interpolation problem. Thus, we have arrived at the following proposition:

Proposition 3.2.4 *With values of τ and l for which* (3.2.1) *is fulfilled, we can construct a bivariate polynomial satisfying the conditions of Theorem 3.2.1 by setting up and solving a system of n homogeneous linear equations in the $(l+1)\big(n - \tau - 1 - \frac{1}{2}(k-1)l\big)$ coefficients of the polynomial.*

If some of the degree bounds of the second requirement of Theorem 3.2.1 are negative, the number of available coefficients is actually greater than the expression used for showing (3.2.1); the inequality might therefore not hold in some cases where the system of equations for finding $Q(x, y)$ could definitely be solved. However, as the degree bounds decrease with increasing j, there would then be a l_0 for which the second requirement gives $\deg(Q_j(x)) < 0$ for all $j > l_0$. But in that case we could set $l = l_0$ and then satisfy (3.2.1). This l_0 is minimum while maximising the number of coefficients with respect to n, τ and k, and can be found simply by looking at the case $j = l$ in the second requirement of Theorem 3.2.1 and requiring this to be non-negative:

$$l = \left\lfloor \frac{n - \tau - 1}{k - 1} \right\rfloor \tag{3.2.2}$$

Put another way, if for some value of τ and the above value of l does not satisfy (3.2.1), then no value of l will satisfy it, and τ has thus crossed the Sudan algorithm's decoding bound for that n and k.

We could analyse further to find the maximum τ which, for given n and k, (3.2.1) can be satisfied, and what the minimum value of l is in that case. We are not going to perform that complete analysis here, but only an abbreviated one, showing the most important result: that the Sudan decoder is only useful for low rate codes. For a full analysis of the bounds, see e.g. the original paper [22].

First of all, setting $l = 1$ in (3.2.1) yields $\tau < \frac{1}{2}(n-k+1)$ which is half the minimum distance. It should come as no surprise that requiring at most one codeword candidate necessitates that we can correct only less than half the minimum distance; beyond this bound, we know that for at least one possible received word, there will be more than one codeword closest. However, it is nice that in this case the algorithm performs maximally well.

The Sudan list-decoding algorithm is much slower than the minimum-distance decoders presented in Chapter 2. It only makes sense to use it if it improves the decoding-capability. Therefore, let us introduce the requirements:

$$\tau \geq \frac{n-k+1}{2} \qquad \text{and} \qquad l \geq 2$$

Using both in (3.2.2) we get

$$2 \leq \left\lfloor \frac{n - \frac{1}{2}(n-k+1) - 1}{k-1} \right\rfloor \qquad \Longleftrightarrow$$
$$2 \leq \left\lfloor \frac{1}{2}\left(\frac{n-2}{k-1} + 1\right) \right\rfloor$$

Asymptotically for large n and k, the above is only satisfied when the rate $\frac{k}{n} \leq \frac{1}{3}$. This reveals that the Sudan algorithm only decodes beyond half the minimum distance for these low-rate codes. As we are often interested in higher-rate codes, this is a severe hindrance to the usability of the algorithm; luckily, it is overcome with the Guruswami-Sudan algorithm, presented in the following section.

3.3 Enters: Guruswami

The Guruswami-Sudan algorithm is an elegant generalisation of the Sudan algorithm which can decode more than half the minimum distance over all rates and almost all choices of n and k. It looks very much like the Sudan algorithm, so its presentation is closely mirrored. The algorithm has in addition to the parameters τ and l, a multiplicity parameter s, and the restrictions for choosing parameters are therefore even more complicated than for the Sudan algorithm.

The Guruswami-Sudan algorithm works almost exactly like the Sudan algorithm, and even solves the polynomial interpolation problem by finding a bivariate polynomial whose factors are the solution to the problem. Like the Sudan algorithm was backed by Theorem 3.2.1, so is the Guruswami-Sudan algorithm backed by a theorem describing what requirements on such a bivariate polynomial are sufficient for it to contain these factors.

Before we can show this theorem, we have a necessary definition:

Definition 3.3.1 *The bivariate polynomial $p(x,y)$ has a zero (a,b) of multiplicity s if and only if all terms of degree less than s in $p(x+a,y+b)$ is zero, and s is the largest such number.*

Now we can present the theorem, which for $s=1$ degenerates to Theorem 3.2.1:

Theorem 3.3.2 *For any instance $(n, k, \tau, [(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})])$ of Problem 3.1.1 in some field $\mathbb{F}$, and for some multiplicity $s \geq 1$ and list size $l \geq 1$, consider a non-zero bivariate polynomial $Q(x, y) = \sum_{j=0}^{l} y^j Q_j(x)$ with coefficients from $\mathbb{F}$, that satisfies*

1. *(x_i, y_i) is a zero of $Q(x, y)$ with multiplicity at least s for $i = 0, \ldots, n-1$*
2. *$\deg(Q_j(x)) \leq s(n-\tau) - 1 - j(k-1)$ for $j = 0, \ldots, l$*

Any polynomial $f(x)$ among the solution of the problem instance satisfies

$$(y - f(x)) \mid Q(x, y)$$

Proof We first prove that if $f(x_i) = y_i$, then $(x - x_i)^s \mid Q(x, f(x))$: From the first requirement, we have that $Q(x + x_i, y + y_i)$ has no terms of degree less than s. Let $p(x) = f(x + x_i) - y_i$; as $p(0) = 0$ we have $x \mid p(x)$, so the univariate polynomial $Q(x + x_i, p(x) + y_i)$ can have no terms of degree less than s; hence

$$x^s \mid Q(x+x_i, p(x)+y_i) \iff x^s \mid Q(x+x_i, f(x+x_i)) \iff (x-x_i)^s \mid Q(x, f(x))$$

This must be true for at least $n-\tau$ of the points (x_i, y_i), which means that $Q(x, f(x))$ is divisible by a polynomial of degree at least $s(n-\tau)$. Now, $Q(x, f(x))$ is the sum of terms of the form $Q_j(x)f(x)^j$, so because $\deg(f(x)) \leq k-1$, the second requirement gives us that each of these has degree at most $s(n-\tau)-1$. Therefore $Q(x, f(x))$ has degree at most $s(n-\tau)-1$. But if it is divisible by a polynomial of higher degree, we must have $Q(x, f(x)) = 0$. As for the proof of Theorem 3.2.1, this implies the sought result. □

As for the Sudan algorithm before, let us immediately specialise this to the decoding setting with the following corollary:

Corollary 3.3.3 *Regard the decoding problem in the setting of polynomial interpolation by using the instance $(n, k, \tau, [(\alpha^0, r_0), \ldots, (\alpha^{n-1}, r_{n-1})])$ of Problem 3.1.1, and for this, let $Q(x, y)$ be a bivariate polynomial satisfying the conditions of Theorem 3.3.2. If $\epsilon \leq \tau$ then $Q(x, y)$ satisfies $(y - f(x)) \mid Q(x, y)$.*

The Guruswami-Sudan algorithm then runs exactly like the Sudan algorithm 3.2, except that the third parameter s needs to be chosen and the constructed polynomial should satisfy Theorem 3.3.2:

Algorithm 3.3.4 (Guruswami-Sudan list decoding)

1. Choose a desired value for the error-correction capacity τ and accompanying appropriate values for the list size l and the multiplicity s.
2. Construct a $Q(x, y)$ satisfying the conditions of Corollary 3.3.3.

3. Find all factors of $Q(x, y)$ of the form $(y - f(x))$ where $\deg(f(x)) < k$.
4. Return as equally likely information words all of the $f(x)$ whose corresponding codeword has least distance to r and less than τ. If no such word exists, declare a decoding failure.

■

Again, we need to show what an acceptable value for τ is as well as appropriate values for s, l, and we need to find a way to construct such a $Q(x, y)$. The remainder of the section is devoted to that. As with the Sudan algorithm, the factoring in step 3 can be done in polynomial time, but we will not discuss such an algorithm.

For constructing a satisfactory $Q(x, y)$ we will employ the same strategy as before: deriving restrictions which are strong enough to always ensure that solving an induced system of homogeneous linear equations will yield the coefficients of $Q(x, y)$.

The first requirement of Theorem 3.3.2 can still be seen as a system of linear equations in the coefficients. If we let $Q(x, y) = \sum_{i,j} q_{i,j} x^i y^j$, then from the definition of zero of multiplicity, we look at the polynomial

$$Q(x + a, y + b) = \sum_{i,j} q_{i,j}(x + a)^i (y + b)^j,$$

which, after some rewriting, gives

$$Q(x + a, y + b) = \sum_{i,j} \tilde{q}_{i,j} x^i y^j, \qquad \text{where}$$

$$\tilde{q}_{i,j} = \sum_{\hat{i} \geq i} \sum_{\hat{j} \geq j} \binom{\hat{i}}{i} \binom{\hat{j}}{j} q_{a,b} a^{\hat{i}-i} b^{\hat{j}-j}$$

For each i, if (x_i, y_i) is a zero of multiplicity s, it means that all $\tilde{q}_{a,b} = 0$ for $a+b < s$; from the above, we see that this specifies $\frac{1}{2}s(s + 1)$ homogeneous linear equations in the coefficients of $Q(x, y)$. Thus, the first requirement of Theorem 3.3.2 can be satisfied by solving in all $\frac{1}{2}ns(s+1)$ such equations. As before, we can definitely find a non-zero solution to this system of equations if the number of variables is greater than the number of equations. The number of variables is the number of coefficients we are allowed, and, by the second requirement of Theorem 3.3.2, this is

$$\sum_{j=0}^{l} \Big(s(n - \tau) - 1 - j(k - 1) + 1\Big) = (l + 1)\big(s(n - \tau) - \tfrac{1}{2}(k - 1)l\big)$$

And so, we get the following requirement:

$$\begin{aligned} \frac{ns(s + 1)}{2} &< (l + 1)\big(s(n - \tau) - \tfrac{1}{2}(k - 1)l\big) \qquad \iff \\ \tau &< n - \tfrac{1}{2}n\frac{s + 1}{l + 1} - \tfrac{1}{2}(k - 1)\frac{l}{s} \end{aligned} \tag{3.3.1}$$

As before, we also have to ensure that the number of coefficients is not zero; that is, at least one of the degree bounds of the second requirement of 3.3.2 should be non-negative. Again, this is satisfied least restrictively by looking at the case $j = 0$, and gives the bound $n > \tau$, but that is already required by the polynomial interpolation problem.

Thus, we have arrived at the following proposition, generalising Proposition 3.2.4:

Proposition 3.3.5 *With values of τ, l and s for which* (3.3.1) *is fulfilled, we can construct a bivariate polynomial satisfying the conditions of Theorem 3.3.2 by setting up and solving a system of $ns(s+1)$ homogeneous linear equations in the $(l+1)\big(s(n-\tau) - \frac{1}{2}(k-1)l\big)$ coefficients of the polynomial.*

Exactly as before, if some of the degree bounds of the second requirement of Theorem 3.3.2 are negative, the bound (3.3.1) is actually too restrictive. This could be amended by choosing the minimum value for l such that the number of coefficients is still maximal, by making sure none of the degree bounds are negative. We can find that l by requiring that the bound for $Q_l(x)$ is non-negative; this gives

$$l = \left\lfloor \frac{s(n-\tau) - 1}{k-1} \right\rfloor$$

Again, if for some value of τ and the above value of l does not satisfy (3.3.1), then no value of l will satisfy it, and τ has thus crossed the Guruswami-Sudan algorithm's decoding bound for that n and k.

We would like to find out precisely what the maximum possible choice of τ is given n and k, and what the necessary values for l and s are in this case. Because of discrete imprecisions, this analysis is very tedious, and so, we will not do it here. However, looking at the asymptotic behaviour when n and k grows large makes it much easier; we will use this tactic to show that for large enough n and k, the Guruswami-Sudan algorithm can decode beyond half the minimum distance for any rate. For a more thorough and precise analysis; see e.g. the original paper [10].

So we let n and k go towards infinity but keeping the rate $\frac{k}{n} = R$ constant. Looking at the bound (3.3.1), it would seem that the fraction $\frac{s}{l}$ plays an important role; as we are not interesting in running time but only maximising τ, we can then also let l and s go towards infinity but keeping their ratio fixed; some functional analysis would reveal that setting $\frac{s}{l} = \sqrt{R}$ might be interesting. Doing that, we get from (3.3.1)

$$\begin{aligned}\frac{\tau}{n} &< 1 - \frac{s+1}{2(l+1)} - \frac{k-1}{2n} \cdot \frac{l}{s} \implies \\ &\to 1 - \tfrac{1}{2}\sqrt{R} - \tfrac{1}{2}R\frac{1}{\sqrt{R}} = 1 - \sqrt{R}\end{aligned}$$

Thus, asymptotically, we can get an error-correction capability of $n(1-\sqrt{R}) = n-\sqrt{nk}$. Comparing this to half the minimum distance, we easily have

$$\begin{aligned} n-\sqrt{nk} &> \frac{n-k}{2} && \iff \\ \frac{(n+k)^2}{4} &> nk && \iff \\ (n-k)^2 &> 0, \end{aligned}$$

which is always true. Thus, for large n and k, the Guruswami-Sudan algorithm can for all rates improve minimum distance decoding. Asymptotically, the bound reached here equals the Johnson bound $J = n-\sqrt{n(k-1)}$ mentioned in Section 1.2.2. A non-asymptotic analysis would show that the Guruswami-Sudan algorithm only requires the slightly weaker $\tau < J$ as well as $\tau < n-k$.

This concludes our presentation of the Sudan and Guruswami-Sudan list decoders. As we will see in the following chapter, a step in Wu's decoding algorithm uses a generalised version of Theorem 3.3.2.

Chapter 4

Wu's combination

Recently, Wu presented in [24] a new approach to list decoding Reed-Solomon codes; the first one markedly different from that of the Guruswami-Sudan algorithm. The algorithm utilises that, even though the Berlekamp-Massey decoder cannot decode when $\epsilon > t$, running the BMA can still reveal important information about the error-locator $\Lambda(x)$. This information is used to reduce the problem of finding the error-locator to that of finding *rational expressions* – i.e. the fraction of two polynomials – of certain degrees that go through some subset of a set of points. This problem sounds familiar, and indeed, Wu solves it using a generalisation of Theorem 3.3.2, which provided the foundation for the Guruswami-Sudan algorithm.

The crucial observation of the approach is that the result $A(x)$ of any run of the BMA can be written as a polynomial combination of any iteration i's intermediate results $A^{(i)}(x)$ and $B^{(i)}(x)$; more precisely, there exists polynomials $a^{(i)}(x)$ and $b^{(i)}(x)$ such that

$$A(x) = A^{(i)}(x)a^{(i)}(x) + xB^{(i)}(x)b^{(i)}(x)$$

More deeply, we can find strict upper bounds for the degrees of $a^{(i)}(x)$ and $b^{(i)}(x)$.

It has long been known, that if we have the error spectrum E, we can run the BMA on it twice and get the error-locator $\Lambda(x)$ – no matter how many errors there are. But the syndromes S are the first $n-k$ elements of E; therefore, if we run the BMA on S, the result can be seen as the $n-k$'th iteration's intermediate result of running the BMA on E. Combined with the above, this means that if running the BMA on S returns $A(x)$ and $B(x)$, there exists polynomials $a(x)$ and $b(x)$ such that

$$\Lambda(x) = A(x)a(x) + xB(x)b(x)$$

And we have upper bounds on the degrees of $a(x)$ and $b(x)$. This equation is key to Wu's algorithm.

Now for the next good idea. We know that for all ϵ error-locations ℓ_i, we have $\Lambda(\alpha^{-\ell_i}) = 0$, because that is how we defined the error-locator. Therefore, for at least those ϵ powers of α, the above equation ensures that the following is true:

$$\frac{b(x)}{a(x)} = -\frac{A(x)}{xB(x)}$$

Put another way, the rational expression $\frac{b(x)}{a(x)}$ goes through at least ϵ of the points $(\alpha^0, \beta_0), \ldots, (\alpha^{n-1}, \beta_{n-1})$, where β_i is the evaluation of the right-hand side of the above equation with $x = \alpha^i$.

Therefore, if we can find *all* rational expressions which go through at least ϵ of the described points, then we know that $\frac{b(x)}{a(x)}$ must be among them. This is a generalisation of the polynomial interpolation problem 3.1.1; it turns out that if we restrict ourselves to only those rational expressions satisfying the bounds on $a(x)$ and $b(x)$, we can generalise Theorem 3.3.2 and solve this problem similarly to how we solve the polynomial interpolation problem in the Guruswami-Sudan algorithm.

In the course of this chapter, we will add all the details, explain further and prove all of the above postulates. However, before beginning this rather long and arduous journey, we give a concise overview of the entire algorithm, followed by an overview of the chapter:

Algorithm 4.0.1 (Overview of Wu's list decoding)

1. Run the BMA on S, which returns $A(x), B(x)$ and L.

2. If $\epsilon \leq t$, we can perform decoding as in the Berlekamp-Massey decoder.

3. Otherwise, we know there exists polynomials $a(x)$ and $b(x)$ of limited degree such that

   $$\Lambda(x) = A(x)a(x) + xB(x)b(x)$$

   Furthermore, we can find a set of n points where we know that the rational expression $\frac{b(x)}{a(x)}$ must go through at least ϵ of them.

   Therefore, we find *all* rational expressions upholding the degree limits which $a(x)$ and $b(x)$ do, and which go through at least $t+1$ of these points, and we know $\frac{b(x)}{a(x)}$ must be among them.

4. These rational expressions lead to a number of possible error-locators per the equation in step 3. This leads to a number of possible codewords, and we select the ones closest to r.

■

In order to prove the existence and degree bounds on the polynomials $a(x)$ and $b(x)$, we have to reinvestigate the BMA and extract a body of properties on its intermediate results. This is done in Section 4.1, and consists only of general results on linear recursions. We then specialise those properties for the coding theory setting in Section 4.2, to extract exactly what we know when having run the BMA on the syndromes.

To then find $a(x)$ and $b(x)$ as rational expressions, we first define and solve the rational interpolation problem in general; this is done in Section 4.3, and is again completely devoid of coding theory. We then return to decoding in Section 4.4 and formally show how the search for $a(x)$ and $b(x)$ can be modelled as the rational interpolation problem and solved using the found results.

We then finally have all the results we need to collect and prove the entire Wu's algorithm, which we do in Section 4.5. However, in the course of applying the rational interpolation to the decoding problem, we identify an anomalous case of the decoding parameter's, where the general approach does not work. In this case, the general Wu's algorithm degenerates to a simpler version, which we will show in Section 4.5.1.

Lastly, in Section 4.6, we discuss various aspects of the algorithm and the derivation.

4.1 BMA revisited

In Section 2.4, we presented the BMA as a method for solving linear recursions. This section will be a continuation of that, where we derive a number of properties that hold for the intermediate values from each iteration of the algorithm. We will once again temporarily forget about coding theory, and the results of this section deal with linear recursions in general. We will extensively discuss the BMA's steps and progression, so the reader might want to keep a bookmark on its description on page 26.

The BMA calculates in each iteration i three important intermediate results: the two new polynomials $A^{(i)}(x)$ and $B^{(i)}(x)$ as well as a linear complexity $L^{(i)}$. At the last iteration m, $(A^{(m)}(x), L^{(m)})$ is a shortest linear recursion producing the input sequence. First, we will show a range of properties that hold for these three series of values. In Section 4.1.1, we will then show that for each of these triples, we can associate a pair of polynomials – the completion-pair – which have similar properties.

Proposition 4.1.1 *If the BMA is run on input $V_1, \dots, V_m$, then the following holds for each iteration i:*

1. *$(A^{(i)}(x), L^{(i)})$ is a shortest linear recursion producing $V_1, \dots, V_i$.*
2. *$L^{(i)} = \mathcal{L}(V_1, \dots, V_i) \leq i$*

3. *The constant term of $A^{(i)}(x)$ is 1.*

4. $\deg(A^{(i)}(x)) \leq L^{(i)}$

5. $\deg(B^{(i)}(x)) \leq i - L^{(i)}$

6. *Equality holds for at least one of Property 4 and 5.*

7. $\gcd(A^{(i)}(x), B^{(i)}(x)) = 1$

In particular, these hold for the m'th iteration and therefore the return values.

Proof
PROPERTY 1: Follows from how the correctness of the BMA: if we run the BMA on the sequence $V_1, \ldots, V_i$, the BMA will return exactly $(A^{(i)}(x), L^{(i)})$ as a shortest linear recursion producing the input.

PROPERTIES 2, 3 AND 4: Follows directly from Property 1.

PROPERTY 5: We do the proof by induction on the iteration of the algorithm. For the base case of iteration zero, we have $L^{(0)} = 0$ and $B^{(0)}(x) = 0$, which satisfies the requirement[1]. For the inductive step, assume now that it holds for iteration $i-1$, and we will prove that this implies iteration i. We consider three cases:

- $\delta^{(i)} = 0$: The BMA sets $B^{(i)}(x) = xB^{(i-1)}(x)$, so using the induction hypothesis, we immediately see that the inequality holds.

- $\delta^{(i)} \neq 0$ and $L^{(i)} = L^{(i-1)}$: The BMA sets $B^{(i)}(x) = xB^{(i-1)}(x)$ so the property holds, as

$$\deg(B^{(i)}(x)) = 1 + \deg(B^{(i-1)}(x)) \leq 1 + (i-1) - L^{(i-1)} = i - L^{(i)}$$

- $\delta^{(i)} \neq 0$ and $L^{(i)} = i - L^{(i-1)}$: The BMA sets $B^{(i)}(x) = \frac{1}{\delta^{(i)}} A^{(i-1)}(x)$, so

$$\deg(B^{(i)}(x)) = \deg(A^{(i-1)}(x)) \leq L^{(i-1)} = i - L^{(i)}$$

PROPERTY 6: Again using induction. The base case holds as $A^{(0)}(x) = 1$ and $L = 0$. For the inductive step, we have two cases depending on which of the equalities hold for the previous step as the induction hypothesis, and independently, we have three cases for how the algorithm progresses in the i'th iteration. This gives six cases and in each case we need to prove that the assumptions imply either $\deg(A^{(i)}(x)) = L^{(i)}$ or $\deg(B^{(i)}(x)) = i - L^{(i)}$.

First, the three cases where the induction hypothesis is $\deg(A^{(i-1)}(x)) = L^{(i-1)}$.

[1]We define $\deg(0) = -\infty$.

- $\deg(A^{(i-1)}(x)) = L^{(i-1)}$ and $\delta^{(i)} = 0$: The BMA will set $L^{(i)} = L^{(i-1)}$ and $A^{(i)}(x) = A^{(i-1)}(x)$, which with the induction hypothesis gives $\deg(A^{(i)}(x)) = L^{(i)}$.
- $\deg(A^{(i-1)}(x)) = L^{(i-1)}$, $\delta^{(i)} \neq 0$ and $L^{(i)} = L^{(i-1)} \geq i - L^{(i-1)}$: If $\deg(B^{(i)}(x)) = i - L^{(i)}$ we are done, so assume $\deg(B^{(i)}(x)) < i - L^{(i)}$, and we will prove $\deg A^{(i)}(x) = \deg A^{(i-1)}(x) = L^{(i)}$. As the BMA sets $A^{(i)}(x) = A^{(i-1)}(x) - \delta^{(i)}xB^{(i-1)}(x)$, this is true as long as the degree of $B^{(i-1)}(x)$ is lower than $L^{(i)} - 1$. But the BMA sets $B^{(i)}(x) = xB^{(i-1)}(x)$, and so $\deg(B^{(i-1)}(x)) < i - 1 - L^{(i-1)} \leq L^{(i)} - 1$.
- $\deg(A^{(i-1)}(x)) = L^{(i-1)}$, $\delta^{(i)} \neq 0$ and $L^{(i)} = i - L^{(i-1)} > L^{(i-1)}$: The BMA sets $B^{(i)}(x) = \frac{1}{\delta^{(i)}}A^{(i-1)}(x)$, which immediately gives
$$\deg(B^{(i)}(x)) = \deg(A^{(i-1)}(x)) = L^{(i-1)} = i - L^{(i)}$$

Next we consider the three cases where the induction hypothesis is $\deg(B^{(i-1)}(x) = i - 1 - L^{(i-1)}$:

- $\deg(B^{(i-1)}(x)) = i - 1 - L^{(i-1)}$ and $\delta^{(i)} = 0$: The BMA sets $B^{(i)}(x) = xB^{(i-1)}(x)$ and $L^{(i)} = L^{(i-1)}$ which immediately implies $\deg(B^{(i)}(x)) = i - L^{(i)}$.
- $\deg(B^{(i-1)}(x)) = i - 1 - L^{(i-1)}$, $\delta^{(i)} \neq 0$ and $L^{(i)} = L^{(i-1)}$: The BMA sets $B^{(i)}(x) = xB^{(i-1)}(x)$ which immediately implies $\deg(B^{(i)}(x)) = i - L^{(i)}$.
- $\deg(B^{(i-1)}(x)) = i - 1 - L^{(i-1)}$, $\delta^{(i)} \neq 0$ and $L^{(i)} = i - L^{(i-1)} > L^{(i-1)}$:
If $\deg(B^{(i)}(x)) = i - L^{(i)}$ we are done, so assume $\deg(B^{(i)}(x)) < i - L^{(i)}$, and we will prove $\deg(A^{(i)}(x)) = L^{(i)} = 1 + \deg(B^{(i-1)}(x))$. As the BMA sets $A^{(i)}(x) = A^{(i-1)}(x) - \delta^{(i)}xB^{(i-1)}(x)$, this is true as long as the degree of $A^{(i-1)}(x)$ is lower than $L^{(i)}$. But the BMA sets $B^{(i)}(x) = \frac{1}{\delta^{(i)}}A^{(i-1)}(x)$, and so $\deg(A^{(i-1)}(x)) = \deg(B^{(i)}(x)) < i - L^{(i)} < L^{(i)}$. This finishes the case and thereby the property.

Property 7: By using Corollary 2.4.6 on page 27 repeatedly, we get
$$\begin{aligned}\begin{pmatrix} A^{(i)}(x) \\ B^{(i)}(x) \end{pmatrix} &= \begin{pmatrix} 1 & -\delta^{(i)}x \\ (\delta^{(i)})^{-1}\flat^{(i)} & (1-\flat^{(i)})x \end{pmatrix} \begin{pmatrix} A^{(i-1)}(x) \\ B^{(i-1)}(x) \end{pmatrix} = \cdots \\ &= \prod_{j=0}^{i-1} \begin{pmatrix} 1 & -\delta^{(i-j)}x \\ (\delta^{(i-j)})^{-1}\flat^{(i-j)} & (1-\flat^{(i-j)})x \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{aligned} \tag{4.1.1}$$

The determinant of each of the product sequences' matrices is directly
$$\begin{aligned}&(1-\flat^{(m-j)})x - (-\delta^{(m-j)}x)(\delta^{(m-j)})^{-1}\flat^{(m-j)} = \\ &\quad x\big(1 - \flat^{(m-j)} + \flat^{(m-j)}\delta^{(m-j)}(\delta^{(m-j)})^{-1}\big) = x\end{aligned}$$

The last equality follows from $(\delta^{(m-j)})^{-1}$ being defined when $\flat^{(i)} = 1$ and otherwise, that entire term is zero. Therefore, the determinant of the entire product sequence equals x^i.

On the other hand, from (4.1.1), we can let

$$\prod_{j=0}^{i-1} \begin{pmatrix} 1 & -\delta^{(i-j)}x \\ (\delta^{(i-j)})^{-1}\flat^{(i-j)} & (1-\flat^{(i-j)})x \end{pmatrix} = \begin{pmatrix} A^{(i)}(x) & p(x) \\ B^{(i)}(x) & q(x) \end{pmatrix},$$

for some polynomials $p(x)$ and $q(x)$, from which we immediately get

$$x^i = \left| \begin{pmatrix} A^{(i)}(x) & p(x) \\ B^{(i)}(x) & q(x) \end{pmatrix} \right| = A^{(i)}(x)q(x) - B^{(i)}(x)p(x)$$

In other words, some polynomial combination of $A^{(i)}(x)$ and $B^{(i)}(x)$ gives x^i. But we know that the $\gcd(A^{(i)}(x), B^{(i)}(x))$ must divide any polynomial combination of them, so it must divide x^i, and hence be a power of x. But as $x \nmid A^{(i)}(x)$ by Property 3 of this proposition, the only power of x to divide $A^{(i)}(x)$ is $x^0 = 1$. Therefore, $\gcd(A^{(i)}(x), B^{(i)}(x)) = 1$. □

4.1.1 Completion-pairs

Each iteration i of the BMA sets $A^{(i)}(x)$ and $B^{(i)}(x)$ to some polynomial combination of the previous $A^{(i-1)}(x)$ and $B^{(i-1)}(x)$. Applying this repeatedly, we see that the final result $A(x)$ can be written as some polynomial combination of any iteration i's intermediate results $A^{(i)}(x)$ and $B^{(i)}(x)$. In this section we formalise these accompanying polynomials' existence and show a series of properties they have.

Let m be the length of the input to a call of the BMA and $i \leq m$ some iteration which we are looking at. The above hand-waving argument can then be made more precise by using Corollary 2.4.6 on page 27 repeatedly:

$$\begin{aligned} \begin{pmatrix} A^{(m)}(x) \\ B^{(m)}(x) \end{pmatrix} &= \begin{pmatrix} 1 & -\delta^{(m)}x \\ (\delta^{(m)})^{-1}\flat^{(m)} & (1-\flat^{(m)})x \end{pmatrix} \begin{pmatrix} A^{(m-1)}(x) \\ B^{(m-1)}(x) \end{pmatrix} = \cdots \\ &= \prod_{j=0}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-j)}x \\ (\delta^{(m-j)})^{-1}\flat^{(m-j)} & (1-\flat^{(m-j)})x \end{pmatrix} \begin{pmatrix} A^{(i)}(x) \\ B^{(i)}(x) \end{pmatrix} \end{aligned}$$

With $A(x) = A^{(m)}(x)$ and selection, we get

$$A(x) = \left((1,0) \cdot \prod_{j=0}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-j)}x \\ (\delta^{(m-j)})^{-1}\flat^{(m-j)} & (1-\flat^{(m-j)})x \end{pmatrix} \right) \begin{pmatrix} A^{(i)}(x) \\ B^{(i)}(x) \end{pmatrix}, \tag{4.1.2}$$

And we see that $A(x)$ can be written as a polynomial combination of $A^{(i)}(x)$ and $B^{(i)}(x)$ with the coefficients being the two polynomials resulting from evaluating the expression in the large parenthesis. Let us introduce a nomenclature for this:

Definition 4.1.2 *Let the BMA be run on some input of length m. The i'th* completion-pair*, $(a^{(i)}(x), b^{(i)}(x))$ is then given by the equation*

$$(a^{(i)}(x), xb^{(i)}(x)) = (1,0) \cdot \prod_{j=0}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-j)}x \\ (\delta^{(m-j)})^{-1}\flat^{(m-j)} & (1-\flat^{(m-j)})x \end{pmatrix}$$

The definition of $b^{(i)}(x)$ might seem a bit odd. That x will always divide the second component of the evaluation of the right-hand side is not too hard to see, as x divides each element in the right row of every matrix of the product sequence. Therefore, in order to reveal this fact in uses of $b^{(i)}(x)$, it makes sense to define it with x divided out.

Now we can precisely describe the main property that we sought to begin with:

Proposition 4.1.3 *For a run of the BMA which returns $A(x)$, the i'th completion-pair $(a^{(i)}(x), b^{(i)}(x))$ for any iteration i satisfies*

$$A(x) = A^{(i)}(x)a^{(i)}(x) + xB^{(i)}(x)b^{(i)}(x)$$

Proof Follows from the definition of completion-pair and (4.1.2) □

This was the easy part. Now we wish to study other properties that these completion-pair polynomials might satisfy. The most important – and the most tediously proved – is bounding their degree:

Proposition 4.1.4 *If the BMA for some input returns $(L, A(x))$, then for all iterations i, the i'th completion-pair $(a^{(i)}(x), b^{(i)}(x))$ satisfies*

$$\deg(a^{(i)}(x)) \le L - L^{(i)} \quad \text{and}$$
$$\deg(b^{(i)}(x)) \le L - (i - L^{(i)}) - 1,$$

Proof Let the input to the algorithm be of length $m \ge i$. We prove the theorem by proving the more general statement

For every $j = i, \dots, m$, the four polynomials $a_A^{(i,j)}(x)$, $b_A^{(i,j)}(x)$, $a_B^{(i,j)}(x)$, $a_B^{(i,j)}(x)$ defined by the equation

$$\begin{pmatrix} a_A^{(i,j)}(x) & xb_A^{(i,j)}(x) \\ a_B^{(i,j)}(x) & b_B^{(i,j)}(x) \end{pmatrix} = \prod_{h=m-j}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-h)}x \\ (\delta^{(m-h)})^{-1}\flat^{(m-h)} & (1-\flat^{(m-h)})x \end{pmatrix}$$

satisfy

1. $\deg(a_A^{(i,j)}(x)) \le L^{(j)} - L^{(i)}$
2. $\deg(b_A^{(i,j)}(x)) \le L^{(j)} - (i - L^{(i)}) - 1$

3. $\deg(a_B^{(i,j)}(x)) \leq j - L^{(j)} - L^{(i)}$

4. $\deg(b_B^{(i,j)}(x)) \leq j - L^{(j)} - (i - L^{(i)})$

We proceed by induction on j. For the base case $j = i$, the empty product sequence becomes the unit matrix, and so $(a_A^{(i,j)}(x), b_A^{(i,j)}(x)) = (1, 0)$ and $(a_B^{(i,j)}(x), b_B^{(i,j)}(x)) = (0, 1)$; all the inequalities are seen to hold.

For the inductive step, assume that the statement holds for $j-1$, and we must show that this implies that it holds for j. Note first that

$$\begin{pmatrix} a_A^{(i,j)}(x) & xb_A^{(i,j)}(x) \\ a_B^{(i,j)}(x) & b_B^{(i,j)}(x) \end{pmatrix} = \begin{pmatrix} 1 & -\delta^{(j)}x \\ (\delta^{(j)})^{-1}\flat^{(j)} & (1-\flat^{(j)})x \end{pmatrix} \begin{pmatrix} a_A^{(i,j-1)}(x) & xb_A^{(i,j-1)}(x) \\ a_B^{(i,j-1)}(x) & b_B^{(i,j-1)}(x) \end{pmatrix} \tag{4.1.3}$$

We consider three cases depending on the progression of the algorithm:

- $\delta^{(j)} = 0$: This implies $\flat^{(j)} = 0$ so (4.1.3) gives

  $$\begin{aligned} (a_A^{(i,j)}(x), b_A^{(i,j)}(x)) &= (a_A^{(i,j-1)}(x), b_A^{(i,j-1)}(x)) \qquad \text{and} \\ (a_B^{(i,j)}(x), b_B^{(i,j)}(x)) &= (xa_B^{(i,j-1)}(x), xb_B^{(i,j-1)}(x)), \end{aligned}$$

  and the degree constraints are easily seen to hold, using the induction hypothesis again.

- $\delta^{(j)} \neq 0$ and $L^{(j)} = L^{(j-1)} \geq j - L^{(j-1)}$: This implies $\flat^{(j)} = 0$ so (4.1.3) gives

  $$\begin{aligned} a_A^{(i,j)}(x) &= a_A^{(i,j-1)}(x) - \delta^{(i)}xa_B^{(i,j-1)}(x) \\ b_A^{(i,j)}(x) &= b_A^{(i,j-1)}(x) - \delta^{(i)}b_B^{(i,j-1)}(x) \\ a_B^{(i,j)}(x) &= xa_B^{(i,j-1)}(x) \\ b_B^{(i,j)}(x) &= xb_B^{(i,j-1)}(x)) \end{aligned}$$

  For $a_B^{(i,j)}(x)$ and $b_B^{(i,j)}(x)$ the degree constraints are easily seen to hold using the induction hypothesis. For the two other polynomials, using $L^{(j)} = \max(L^{(j-1)},\ j - L^{(j-1)})$, we have:

  $$\begin{aligned} \deg(a_A^{(i,j)}(x)) &\leq \max(\deg(a_A^{(i,j-1)}(x)),\ 1 + \deg(a_B^{(i,j-1)}(x))) \\ &\leq \max(L^{(j-1)} - L^{(i)},\ 1 + (j - 1 - L^{(j-1)} - L^{(i)})) \\ &= L^{(j)} - L^{(i)} \\ \deg(b_A^{(i,j)}(x)) &\leq \max(\deg(b_A^{(i,j-1)}(x)),\ \deg(b_B^{(i,j-1)}(x))) \\ &\leq \max(L^{(j-1)} - (i - L^{(i)}) - 1,\ j - 1 - L^{(j-1)} - (i - L^{(i)})) \\ &= L^{(j)} - (i - L^{(i)}) - 1 \end{aligned}$$

- $\delta^{(j)} \neq 0$ and $L^{(j)} = j - L^{(j-1)} > L^{(j-1)}$: This implies $\flat^{(j)} = 1$ so (4.1.3) gives

$$\begin{aligned} a_A^{(i,j)}(x) &= a_A^{(i,j-1)}(x) - \delta^{(i)} x a_B^{(i,j-1)}(x) \\ b_A^{(i,j)}(x) &= b_A^{(i,j-1)}(x) - \delta^{(i)} b_B^{(i,j-1)}(x) \\ a_B^{(i,j)}(x) &= \tfrac{1}{\delta^{(j)}} a_A^{(i,j-1)}(x) \\ b_B^{(i,j)}(x) &= \tfrac{1}{\delta^{(j)}} x b_A^{(i,j-1)}(x) \end{aligned}$$

We still have $L^{(j)} = \max(L^{(j-1)},\ j - L^{(j-1)})$, so the degree constraints for $a_A^{(i,j)}(x)$ and $b_A^{(i,j)}(x)$ hold as before. For the two other polynomials, we have

$$\begin{aligned} \deg(a_B^{(i,j)}(x)) &= \deg(a_A^{(i,j-1)}(x)) \\ &\leq L^{(j-1)} - L^{(i)} \\ &< L^{(j)} - L^{(i)} \\ \deg(b_B^{(i,j)}(x)) &= 1 + \deg(b_A^{(i,j-1)}(x)) \\ &\leq 1 + L^{(j-1)} - (i - L^{(i)}) - 1 \\ &= j - L^{(j)} - (i - L^{(i)}) \end{aligned}$$

This concludes all three cases and thereby the proof of the general statement. This statement directly implies the proposition for $j = m$. □

The next property is a rather obvious but still useful one:

Proposition 4.1.5 *The left polynomial of a completion-pair has constant term 1.*

Proof Let $(a^{(i)}(x), b^{(i)}(x))$ be the i'th completion-pair of a run of the BMA resulting in $A(x)$. By Proposition 4.1.3 we have

$$A(x) = A^{(i)}(x)a^{(i)}(x) + xB^{(i)}(x)b^{(i)}(x)$$

The constant term of $A(x)$ is 1 because it is a linear recursion. The constant term of $A^{(i)}(x)$ is 1 by Proposition 4.1.1 Property 3, and the constant term of $xB^{(i)}(x)b^{(i)}(x)$ is obviously 0. It follows that the constant term of $a^{(i)}(x)$ must be 1. □

We follow with a deeper observation, reminding one of the properties of $A^{(i)}(x)$ and $B^{(i)}(x)$ which we showed in the previous section:

Proposition 4.1.6 *If $(a(x), b(x))$ is a completion-pair, then* $\gcd(a(x), b(x)) = 1$.

Proof Let m be the length of the input to the BMA, let i be some iteration and let $(a^{(i)}(x), b^{(i)}(x))$ be the i'th completion-pair.

We first note that $x \nmid a^{(i)}(x)$ because of Proposition 4.1.5. Now, similar to the proof of Proposition 4.1.1 Property 7, we will now show that the greatest common divisor of $a^{(i)}(x)$ and $b^{(i)}(x)$ is 1, by showing that some polynomial combination of them gives a power of x.

By definition we have

$$(a^{(i)}(x), b^{(i)}(x)) = (1,0) \cdot \prod_{j=0}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-j)}x \\ (\delta^{(m-j)})^{-1}\flat^{(m-j)} & (1-\flat^{(m-j)})x \end{pmatrix}$$

As we did in the proof of Proposition 4.1.1 Property 7, we easily see that the determinant of each of the product sequences' matrices is x, and hence, the determinant of the entire sequence is x^{m-i}. On the other hand, we can write

$$\prod_{j=0}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-j)}x \\ (\delta^{(m-j)})^{-1}\flat^{(m-j)} & (1-\flat^{(m-j)})x \end{pmatrix} = \begin{pmatrix} a^{(i)}(x) & b^{(i)}(x) \\ p(x) & q(x) \end{pmatrix},$$

for some $p(x)$ and $q(x)$. From this, we get

$$x^{m-i} = \left| \prod_{j=0}^{m-i-1} \begin{pmatrix} 1 & -\delta^{(m-j)}x \\ (\delta^{(m-j)})^{-1}\flat^{(m-j)} & (1-\flat^{(m-j)})x \end{pmatrix} \right| = a^{(i)}(x)q(x) - b^{(i)}(x)p(x)$$

That is, some polynomial combination of $a^{(i)}(x)$ and $b^{(i)}(x)$ gives the determinant of the product sequence, i.e. x^{m-j}. But we know that $\gcd(a^{(i)}(x), b^{(i)}(x))$ must divide any polynomial combination of them so it must divide x^{m-j}, and hence, be a power of x. But as $x \nmid a^{(i)}(x)$, the only power of x to divide $a^{(i)}(x)$ is $x^0 = 1$. Therefore, $\gcd(a^{(i)}(x), b^{(i)}(x)) = 1$. □

Corollary 4.1.7 *If $(a(x), 0)$ is a completion-pair, then $a(x) = 1$.*

Last, we can show that the completion-pairs for higher iterations are in some sense unique:

Proposition 4.1.8 *If the BMA returns $(A(x), L)$ on some input of length m, then for all iterations i with $L < i \leq m$, the i'th completion-pair is the only pair of polynomials satisfying for $(a(x), b(x))$ both*

$$A(x) = A^{(i)}(x)a(x) + xB^{(i)}(x)b(x)$$

as well as

$$\deg(a(x)) \leq L - L^{(i)} \qquad \text{and}$$
$$\deg(b(x)) \leq L - (i - L^{(i)}) - 1,$$

Proof From Propositions 4.1.3 and 4.1.4, it is clear that the i'th completion-pair satisfies the constraints. We prove uniqueness by contradiction. Let $(a^{(i)}(x), b^{(i)}(x))$ be the i'th completion-pair, and assume then that there exists a different pair $(\tilde{a}^{(i)}(x), \tilde{b}^{(i)}(x))$ satisfying for $(a(x), b(x))$ the constraints. Then

$$\begin{aligned} A^{(i)}(x)a^{(i)}(x) + xB^{(i)}(x)b^{(i)}(x) = A^{(i)}(x)\tilde{a}^{(i)}(x) + xB^{(i)}(x)\tilde{b}^{(i)}(x) \qquad \iff \\ A^{(i)}(x)\big(a^{(i)}(x) - \tilde{a}^{(i)}(x)\big) = xB^{(i)}(x)\big(\tilde{b}^{(i)}(x) - b^{(i)}(x)\big) \end{aligned}$$

By Proposition 4.1.1 Property 7, $A^{(i)}(x)$ and $B^{(i)}(x)$ are coprime, so the above equation implies $B^{(i)}(x) \mid \big(a^{(i)}(x) - \tilde{a}^{(i)}(x)\big)$. Furthermore, Proposition 4.1.1 Property 3 also gives that $A^{(i)}(x)$ and x are coprime, and we then also reach $A^{(i)}(x) \mid \big(\tilde{b}^{(i)}(x) - b^{(i)}(x)\big)$. These two relations along with the degree constraints of Proposition 4.1.4 necessitate

$$\deg(A^{(i)}(x)) \le \deg\left(\tilde{b}^{(i)}(x) - b^{(i)}(x)\right) \le L - (i - L^{(i)}) - 1 \qquad \text{and} \tag{4.1.4}$$

$$\deg(B^{(i)}(x)) \le \deg\left(a^{(i)}(x) - \tilde{a}^{(i)}(x)\right) \le L - L^{(i)} \tag{4.1.5}$$

From Proposition 4.1.1 Property 6, we have either

$$\deg(A^{(i)}(x)) = L^{(i)} \qquad \text{or} \qquad \deg(B^{(i)}(x)) = i - L^{(i)}$$

In the first case, (4.1.4) yields

$$L^{(i)} \le L - (i - L^{(i)}) - 1 \iff L \ge i + 1$$

In the second case, (4.1.5) yields

$$i - L^{(i)} \le L - L^{(i)} \iff L \ge i$$

In both cases, the proposition's assumption $L < i$ is contradicted. □

This concludes our lengthy study of the intermediate values of the BMA. With these results, we need no further investigation of the algorithm itself and its progression; we don't even need the exact definition of the completion-pairs. We can draw solely on the propositions of this section for the properties we will need in order to prove the correctness of Wu's algorithm.

4.2 BMA as a stepping stone towards list decoding

We will now return to coding theory. In the presentation of the Berlekamp-Massey decoder in Section 2.5, running the BMA on the syndromes S returns the error-locator $\Lambda(x)$ whenever $\epsilon \le t$. However, when $\epsilon > t$, we completely give up. But now, with the new theory about the BMA in hand, we can use the concept of completion-pairs to reveal important relations that $\Lambda(x)$ has with the result of the BMA in these cases.

First we need a generalisation of Theorem 2.5.1 on page 27. It would be nice if $(\Lambda(x), \epsilon)$ was simply the shortest linear recursion producing E, but unfortunately, we cannot be sure of that. However, it is the shortest linear recursion *cyclically* producing E; that is, $(\Lambda(x), \epsilon)$ produces the infinite sequence $E_1, E_2, \ldots, E_n, E_1, E_2, \ldots$ Unfortunately, the proof strategy used to prove 2.5.1 will not help us, as it would assume $2\epsilon < n$; the presentation of the theory given here does not easily give a neat proof, as e.g. the angle of Fourier transformations does, done in Blahut [4]. The exact proof of this fact in the chosen notation should not be important, and therefore we will not give such a proof, but instead refer to other presentations, e.g. [3, 4].

Proposition 4.2.1 $(\Lambda(x), \epsilon)$ *is the unique shortest linear recursion cyclically producing* E.

Now, the obvious question is whether any finite run of the BMA will find a linear recursion that will cyclically produce E indefinitely. It will, and this is captured by the following lemma:

Lemma 4.2.2 $(\Lambda(x), \epsilon)$ *is the unique shortest linear recursion producing the sequence* $E_{m(1)}, E_{m(2)}, \ldots, E_{m(2\nu)}$, *where* $m(x) = 1 + (x - 1 \bmod n)$ *and* $\nu \geq \epsilon$.

Proof The sequence $\hat{E} = E_{m(1)}, E_{m(2)}, \ldots, E_{m(2\nu)}$ is the beginning of a cyclic production of E, so by Proposition 4.2.1, $(\Lambda(x), \epsilon)$ produces it. Now assume that another linear recursion A of length $L \leq \epsilon$ produces $\hat{E}$. According to Proposition 2.4.2 on page 22, as $2\nu \geq \epsilon + L$, A will always produce the same as $(\Lambda(x), \epsilon)$. As Proposition 4.2.1 ensures that $(\Lambda(x), \epsilon)$ is the unique shortest linear recursion to do this, no such A can exist. □

Given an upper bound on ϵ, we therefore have an upper bound on the number of iterations to run the BMA in. As we discussed in Section 1.2.2, we can never hope for a polynomial-time list-decoder with a bound higher than $n - k$, so we can safely use that. Of course this is all theoretical, as E is not known to the receiver. However, with the results of previous section on the intermediate values of the BMA, it does give us insights on what happens when running the BMA on S, and we can now present the main result of this section, which provides the first step of Wu's algorithm:

Theorem 4.2.3 *Assume* $\epsilon < n - k$. *If the BMA is run on the syndromes* S *and returns* $(L, A(x), B(x))$, *then there exist uniquely determined polynomials* $a(x)$ *and* $b(x)$ *such that*

$$\Lambda(x) = A(x)a(x) + xB(x)b(x), \tag{4.2.1}$$

as well as

$$\deg(a(x)) \leq \epsilon - L \qquad \text{and}$$
$$\deg(b(x)) \leq \epsilon + L - (n - k + 1) \tag{4.2.2}$$

Proof As $2\epsilon < 2(n-k)$, by Lemma 4.2.2, running the BMA on the sequence $\hat{E} = E_{m(1)}, \dots, E_{m(2n-2k)}$ will result in the linear recursion $(\Lambda(x), \epsilon)$. As $S_i = E_i$ for $i = 1, \dots, n-k$, the $(L, A(x), B(x))$ returned by the BMA when run on S will be the $(n-k)$'th iteration's intermediate results when the BMA is run on $\hat{E}$. But then the $n-k$'th completion-pair $(a(x),\ b(x))$ for this run of the BMA satisfies (4.2.1) by Proposition 4.1.3 on page 45 and the constraints (4.2.2) by Proposition 4.1.4. Uniqueness is furthermore guaranteed by Proposition 4.1.8, as $\epsilon < n-k$. □

Note that the degree bounds for $a(x)$ and $b(x)$ beautifully fit the case of $\epsilon \leq t$: From Section 2.5, we know that then $A(x) = \Lambda(x)$ and $L = \epsilon \leq \lfloor \frac{n-k}{2} \rfloor$; therefore, the solution must be $a(x) = 1$ and $b(x) = 0$. Tightly and neatly, we get the degree bounds $\deg(a(x)) = 0$ and $\deg(b((x)) < 0$.

For the remainder of this chapter, we will let L, $A(x)$ and $B(x)$ be the result of running the BMA on S. Furthermore, we let $a(x)$ and $b(x)$ be the unique pair of polynomials satisfying the above theorem. We can now use the results of the previous section to further specify them:

Proposition 4.2.4 *The polynomials $a(x)$ and $b(x)$ satisfy*

1. *The constant term of $a(x)$ is 1.*
2. *$a(x)$ and $b(x)$ are coprime.*
3. *$b(x) = 0 \implies a(x) = 1$*

Proof
These all follow from the results on completion-pairs that we derived in Section 4.1.1: Property 1 follows from Proposition 4.1.5. Property 2 follows from Proposition 4.1.6. Property 3 follows from Corollary 4.1.7. □

The found results can also be used to limit the possibilities of where codewords may lie; this will be useful when we put together Wu's algorithm in Section 4.5. We have two such results. We begin with the most obvious and intuitive one:

Proposition 4.2.5 *If $\epsilon < n-k$ then $L \leq \epsilon$.*

Proof Assume oppositely that $L > \epsilon$. By Theorem 4.2.3, the degree upper-bound for $a(x)$ becomes negative, implying $a(x) = 0$. But this contradicts that the constant term of $a(x)$ is 1 by Proposition 4.2.4. □

Proposition 4.2.6 *If $\epsilon \leq n-k-L$, then c is the only codeword with distance at most $n-k-L$ from r, and $(\Lambda(x), \epsilon) = (A(x), L)$.*

Proof Let $\tilde{c}$ be any codeword with distance $\tilde{\epsilon} \leq n-k-L$ from r; as if $\tilde{c}$ was the sent codeword, we can associate an error locator $\tilde{\Lambda}(x)$ with it. As the syndromes only depend on r, the result of running the BMA on it is $(A(x), B(x), L)$. Now, from Theorem 4.2.3, there exist uniquely determined polynomials $\tilde{a}(x)$ and $\tilde{b}(x)$ such that

$$\tilde{\Lambda}(x) = A(x)\tilde{a}(x) + xB(x)\tilde{b}(x), \tag{4.2.3}$$

as well as

$$\deg(\tilde{a}(x)) \leq \tilde{\epsilon} - L \qquad \text{and}$$
$$\deg(\tilde{b}(x)) \leq \tilde{\epsilon} + L - (n-k+1)$$

However, on the degree constraint of $\tilde{b}(x)$, we have

$$\begin{aligned}\deg(\tilde{b}(x)) &\leq \tilde{\epsilon} + L - (n-k+1)\\ &\leq (n-k-L) + L - (n-k+1)\\ &= -1\end{aligned}$$

This implies $\tilde{b}(x) = 0$ which, by Proposition 4.2.4, implies $\tilde{a}(x) = 1$. But this holds for any codeword within distance $n-k-L$ of r, and by (4.2.3), the error-locators would be the same for each of them. This means that there can be only one such codeword, namely c. □

In Wu's algorithm, once the BMA has been run and we have determined that $\epsilon > t$, we attempt to find $\Lambda(x)$ by searching for the $a(x)$ and $b(x)$ polynomials defined here. Because $a(x)$ and $b(x)$ are coprime, we can instead search for the rational expression $\frac{b(x)}{a(x)}$, by solving an appropriate instance of a rational interpolation problem. The properties found for $a(x)$ and $b(x)$ in this section are essential for setting up and solving this problem. But before we can show that, we need to step out of coding theory once again: in the next section, we will formally define this rational interpolation problem, and solve it under certain criteria. It turns out that these criteria are met in the decoding setting whenever $\epsilon < n - \sqrt{n(k-1)}$; the same bound that the Guruswami-Sudan algorithm meets.

4.3 Rational interpolation

Problem 3.1.1 on page 30 was about finding polynomials of up to a certain degree, which go through a number of predefined points. We can look at the same problem, but where we accept *rational expressions* of certain degrees. In this section, we will formalise that problem and solve it when certain criteria are met. We will once again temporarily turn from coding theory, and this section will deal with rational interpolation in general.

A rational expression over some field $\mathbb{F}$ is simply a polynomial fraction $\frac{p(x)}{q(x)}$ where $p(x), q(x) \in F[x]$ and $q(x) \neq 0$. Often, $p(x)$ and $q(x)$ are allowed to share factors, but we will only regard rational expressions where they are coprime. The set of rational expressions form a field over the naturally induced addition and multiplication operations, and this field is usually denoted $\mathbb{F}(x)$. For a more complete description; see e.g. [21].

Like polynomials, a rational expression $\frac{p(x)}{q(x)} \in \mathbb{F}(x)$ can be evaluated at the points of the field $\mathbb{F}$ and it has well-defined values for most input; however, for all $z \in \mathbb{F}$ such that $q(z) = 0$, we say that $\frac{p(x)}{q(x)}$ has a pole and its value is not well-defined (as we require $p(x)$ and $q(x)$ to be coprime, we cannot also have $p(z) = 0$). As is customary notation, we then say that $\frac{p(z)}{q(z)} = \infty$, and that "$\frac{p(x)}{q(x)}$ goes through the point (z, ∞)".

As will become apparent later, our exposition becomes more elegant if we also slightly extend the notation for bivariate polynomials to accommodate for a possible infinite y-value. We recall Definition 3.3.1 on page 34 for zeroes of multiplicity; we want a generalisation of this supporting that some point at infinity (z, ∞) is a zero of multiplicity s of some $p(x, y) \in F[x, y]$. Informally, in a real-valued field, this would mean that $p(z, y)$ approaches zero in the right way for $y \to \infty$, which is the same as saying $p(z, \frac{1}{y})$ approaches zero in the same way for $y \to 0^+$. But $p(z, \frac{1}{y})$ is not a polynomial. This idea can be formalised using rational expressions in projective space, which can also give the idea of using homogeneous polynomials; this yields the so-called companion polynomial which turns out to give a good generalisation:

Definition 4.3.1 *The bivariate polynomial $p(x, y)$ has a zero (z, ∞) of multiplicity s if and only if the companion polynomial $\bar{p}(x, y) = y^{d_y} p(x, \frac{1}{y})$ has a zero $(z, 0)$ of multiplicity s, where d_y is the y-degree of $p(x, y)$.*

Now we can define the rational interpolation problem in a way closely mirroring that of polynomial interpolation 3.1.1:

Problem 4.3.2 (Rational interpolation) *Given positive integers n, k_1, k_2 and $\hat{\tau}$ with $n > \hat{\tau}$, as well as n distinct points $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ where $x_i \in \mathbb{F}$ and $y_i \in \mathbb{F} \cup \{\infty\}$ for all i and some field $\mathbb{F}$, find all rational expressions $\frac{f_1(x)}{f_2(x)} \in \mathbb{F}(x)$, with coprime $f_1(x)$ and $f_2(x)$, $\deg(f_1(x)) < k_1$ and $\deg(f_2(x)) < k_2$, which go through at least $n - \hat{\tau}$ of those points.*

The reader might be baffled by the use of $\hat{\tau}$ instead of simply τ in the above problem description. This inelegant discrepancy is due to the application as a decoding algorithm: for the Sudan and Guruswami-Sudan algorithms, τ as a decoding bound was also the number of points allowed to be wrong in the interpolation, and we could for simplicity and intuition reuse the same symbol. For Wu's algorithm, this relationship is more complex, so we need to use different notation for these two values; we have opted for retaining τ as the decoding bound, and introduced $\hat{\tau}$ for the interpolation parameter.

Wu found in [24] that the Guruswami-Sudan interpolation algorithm can be elegantly generalised to solve the rational interpolation problem, by generalising Theorem 3.3.2 on page 35. Handling the rational expressions and points at infinity makes the proof somewhat more involved, though.

Theorem 4.3.3 *For any instance $\big(n,\ k_1,\ k_2,\ \hat{\tau},\ [(x_0, y_0), \dots, (x_{n-1}, y_{n-1})]\big)$ of Problem 4.3.2 in some field $\mathbb{F}$, and for some multiplicity $s \geq 1$ and list size $l \geq 1$, consider a non-zero bivariate polynomial $Q(x, y) = \sum_{j=0}^{l} y^j Q_j(x)$, with coefficients from $\mathbb{F}$, that satisfies*

1. *(x_i, y_i) is a zero of $Q(x, y)$ with multiplicity at least s for $i = 0, \dots, n-1$*
2. *$\deg(Q_j(x)) \leq s(n - \hat{\tau}) - j(k_1 - k_2) - lk_2 + (l-1)$ for $j = 0, \dots, l$*

Any rational expression $\frac{f_1(x)}{f_2(x)}$ among the solution of the problem instance satisfies

$$(f_2(x)y - f_1(x)) \mid Q(x, y)$$

Proof Let $\frac{f_1(x)}{f_2(x)}$ be a solution to the problem instance, and let $d_y \leq l$ be the y-degree of $Q(x, y)$. Define now

$$\hat{Q}(x) = f_2(x)^{d_y} Q\big(x, \tfrac{f_1(x)}{f_2(x)}\big) = \sum_{j=0}^{d_y} f_1(x)^j f_2(x)^{d_y - j} Q_j(x), \tag{4.3.1}$$

Similarly to how we proved Theorems 3.2.1 and 3.3.2, we first show that $\hat{Q}(x) = 0$ and this will lead to the sought.

To show $\hat{Q}(x) = 0$, we begin by showing that if $\frac{f_1(x_i)}{f_2(x_i)} = y_i$, then $(x - x_i)^s \mid \hat{Q}(x)$; we consider two cases depending on y_i.

- If $y_i \neq \infty$, let $p(x) = \frac{f_1(x+x_i)}{f_2(x+x_i)} - y_i$. Note that $x \mid f_2(x+x_i)p(x)$ as it is a polynomial with $f_2(0 + x_i)p(0) = 0$; so let $f_2(x + x_i)p(x) = x\grave{p}(x)$ for some $\grave{p}(x)$. Now, from the first requirement and the definition of zeroes of multiplicity, we know that the bivariate polynomial $Q(x + x_i, y + y_i)$ has no terms of degree less than s; therefore, the univariate polynomial $Q(x + x_i, p(x) + y_i)$ must have the form

  $$Q(x + x_i, p(x) + y_i) = \sum_{i+j \geq s} x^i p(x)^j a_{i,j},$$

  for some coefficients $a_{i,j}$, where also $j \leq d_y$. This means

  $$\begin{aligned} &f_2(x + x_i)^{d_y} Q(x + x_i, p(x) + y_i) = \\ &\sum_{i+j \geq s} x^i (f_2(x + x_i)p(x))^j f_2(x + x_i)^{d_y - j} a_{i,j} = \\ &\sum_{i+j \geq s} x^{i+j} \grave{p}(x)^j f_2(x + x_i)^{d_y - j} a_{i,j} \end{aligned}$$

But then we immediately have

$$\begin{aligned} x^s &\mid f_2(x+x_i)^{d_y} Q(x+x_i, p(x)+y_i) && \iff \\ x^s &\mid f_2(x+x_i)^{d_y} Q(x+x_i, \tfrac{f_1(x+x_i)}{f_2(x+x_i)}) && \iff \\ (x-x_i)^s &\mid f_2(x)^{d_y} Q(x, \tfrac{f_1(x)}{f_2(x)}) = \hat{Q}(x) \end{aligned}$$

which finishes this first case.

- The other case is where $y_i = \infty$. As $Q(x,y)$ has a zero of multiplicity s at (x_i, y_i), then $\bar{Q}(x,y) = y^{d_y} Q(x, \frac{1}{y})$ has a zero of multiplicity s at $(x_i, 0)$. Therefore $\bar{Q}(x+x_i, y+0)$ has no terms of degree less than s; we have

  $$\bar{Q}(x+x_i, y) = y^{d_y} Q(x+x_i, \tfrac{1}{y}) = \sum_{j=0}^{d_y} y^{d_y-j} Q_j(x+x_i)$$

  For $\bar{Q}(x+x_i, y)$ to have no terms of degree less than s, each of the $y^{d_y-j}Q_j(x+x_i)$ must have this property since they all have different y-degree. Therefore, $Q_j(x)$ has no terms of degree lower than $s-(d_y-j)$, which is interesting whenever this expression is positive; we can therefore write for all $j = 0, \dots, d_y$ that

  $$Q_j(x+x_i) = x^{\backslash s-(d_y-j)/} \grave{Q}_j(x),$$

  for some polynomials $\grave{Q}_j(x)$, and where $\backslash n/ = n[n>0]$.

  We will now rewrite $\hat{Q}(x+x_i)$ and utilise the above equation. As $\frac{f_1(x_i)}{f_2(x_i)} = y_i = \infty$ we have $f_2(x_i) = 0$ and therefore $x \mid f_2(x+x_i) \implies f_2(x+x_i) = x\grave{f}_2(x)$ for some $\grave{f}_2(x)$. Therefore, we can write

  $$\begin{aligned} \hat{Q}(x+x_i) &= \sum_{j=0}^{d_y} f_1(x+x_i)^j f_2(x+x_i)^{d_y-j} Q_j(x+x_i) \\ &= \sum_{j=0}^{d_y} f_1(x+x_i)^j \left(x^{d_y-j}\grave{f}_2(x)^{d_y-j}\right)\left(x^{\backslash s-(d_y-j)/}\grave{Q}_j(x)\right) \\ &= \sum_{j=0}^{d_y} f_1(x+x_i)^j \grave{f}_2(x)^{d_y-j} \grave{Q}_j(x) x^{d_y-j+\backslash s-(d_y-j)/} \end{aligned}$$

  It is easy to see that the exponent on x in each of the last summation's terms is always at least s. Therefore x^s divides each of the summation's terms and therefore the sum $\hat{Q}(x+x_i)$. Therefore $(x-x_i)^s \mid \hat{Q}(x)$. This finishes the other case.

So we have now that $(x-x_i)^s \mid \hat{Q}(x)$ for all i where $\frac{f_1(x_i)}{f_2(x_i)} = y_i$. But as $\frac{f_1(x)}{f_2(x)}$ is a solution to the problem instance, this must hold for at least $n - \hat{\tau}$ values of i. Therefore, $\hat{Q}(x)$ must be divisible by a polynomial of degree at least $s(n-\hat{\tau})$.

The degree of $\hat{Q}(x)$ is less than $s(n-\hat{\tau})$ as each term $\hat{Q}_j(x)$ of the sum in (4.3.1) has

$$\begin{aligned}\deg(\hat{Q}_j(x)) &\le j(k_1-1)+(l-j)(k_2-1)+\deg(Q_j(x))\\ &\le j(k_1-k_2)+lk_2-l+\big(s(n-\hat{\tau})-j(k_1-k_2)-lk_2+(l-1)\big)\\ &= s(n-\hat{\tau})-1\end{aligned}$$

Then $\hat{Q}(x)$ is divisible by a polynomial of higher degree, so we must have $\hat{Q}(x)=0$; as $f_2(x)\neq 0$, it means we have $Q(x,\frac{f_1(x)}{f_2(x)})=0$.

Considering $Q(x,y)$ from the ring $\mathbb{F}(x)[y]$ – that is, as a univariate polynomial in y with coefficients from $\mathbb{F}(x)$ – the expression $\frac{f_1(x)}{f_2(x)}$ is then a zero of $Q(x,y)$; this implies that for some element $q(y)\in\mathbb{F}(x)[y]$, we have

$$Q(x,y)=q(y)\left(y-\tfrac{f_1(x)}{f_2(x)}\right) \tag{4.3.2}$$

From the intuition of Gauss' lemma and related results, we expect that this implies that there exists an element $\grave{q}(y)\in\mathbb{F}[x][y]$ such that

$$Q(x,y)=\grave{q}(y)(f_2(x)y-f_1(x)), \tag{4.3.3}$$

which implies the theorem to be proved. Indeed, it is easy to show: As a univariate polynomial in y, all coefficients of $Q(x,y)$ are elements of $\mathbb{F}[x]$, so (4.3.2) implies that the constant term of $q(y)$ must be in $\mathbb{F}[x]$ and a multiple of $f_2(x)$. Furthermore, with $i>0$, the equation implies that the i'th coefficient of $Q(x,y)$ regarded as a univariate polynomial in y is given by the expression

$$q_{i-1}-\tfrac{f_1(x)}{f_2(x)}q_i,$$

where q_i is the i'th coefficient of $q(y)$. This must be an element of $\mathbb{F}[x]$, so with $q_0\in\mathbb{F}[x]$ and $f_2(x)\mid q_0$, we get inductively for all i that $q_i\in\mathbb{F}[x]$ and $f_2(x)\mid q_i$. But then $q(y)=f_2(x)\grave{q}(y)$ for some $\grave{q}(y)\in\mathbb{F}[x][y]$. This gives (4.3.3). □

As for the two previous theorems 3.2.1 and 3.3.2 on polynomial interpolation, Theorem 4.3.3 only describes properties on a bivariate polynomial satisfying the requirements, but says nothing about whether this polynomial can be constructed or not. However, the requirements are quite similar, calling for a similar construction strategy.

For each point (x_i,y_i) where $y_i\neq\infty$, we saw in Section 3.3 how a polynomial where the first requirement holds can be found by solving a system of $\frac{1}{2}s(s+1)$ homogeneous linear equations in the coefficients. Whenever $y_i=\infty$, the first requirement holds for (x_i,y_i) whenever $(x_i,0)$ is a zero of multiplicity s of the polynomial $\bar{Q}(x,y)=\sum_{j=0}^{d_y}y^{l-j}Q_j(x)$. This can again be achieved by solving a system of $\frac{1}{2}s(s+1)$ homogeneous linear equations in the coefficients, and – as for the Guruswami-Sudan algorithm – we get a total system of $\frac{1}{2}ns(s+1)$ homogeneous linear equations.

This system definitely has non-zero solutions whenever there are more variables than equations. The number of variables is the number of coefficients of $Q(x, y)$ we are allowed, so this is bounded by the second requirement to be at most

$$\begin{aligned}&\sum_{j=0}^{l}\Big(s(n-\hat{\tau})-j(k_1-k_2)-lk_2+(l-1)+1\Big)=\\&(l+1)\big(s(n-\hat{\tau})-lk_2+l\big)-\tfrac{1}{2}l(l+1)(k_1-k_2)=\\&(l+1)\big(s(n-\hat{\tau})-\tfrac{1}{2}l(k_1+k_2-2)\big)\end{aligned}$$

Thus, we can definitely find a solution to the system if we impose

$$\tfrac{1}{2}ns(s+1)<(l+1)\big(s(n-\hat{\tau})-\tfrac{1}{2}l(k_1+k_2-2)\big) \tag{4.3.4}$$

As before, the only additional thing to ensure is that not all of the degree bounds in the second requirement of Theorem 4.3.3 are negative. As we don't know which of k_1 and k_2 is the greatest, however, this is not as trivial as before, and we get two bounds: if $k_1 \le k_2$, the most liberal bound is from the case $j = l$, which gives us

$$k_1 \le k_2 \implies s(n-\hat{\tau})-l(k_1-1)>0 \tag{4.3.5}$$

If $k_1 > k_2$, we get the bound from the case $j = 0$, which gives

$$k_1 > k_2 \implies s(n-\hat{\tau})-l(k_2-1)>0 \tag{4.3.6}$$

Thus, we have arrived at the following proposition, mirroring Proposition 3.3.5:

Proposition 4.3.4 *With values of τ, l and s for which* (4.3.4), (4.3.5) *and* (4.3.6) *are fulfilled, we can construct a bivariate polynomial satisfying the conditions of Theorem 4.3.3 by setting up and solving a system of $ns(s+1)$ homogeneous linear equations in the $(l+1)\big(s(n-\hat{\tau})-\frac{1}{2}l(k_1+k_2-2)\big)$ coefficients of the polynomial.*

As with the Sudan and Guruswami-Sudan algorithms, if some of the degree bounds required in Theorem 4.3.3 are negative, the bound (4.3.4) is actually too restrictive. However, contrarily to those previous cases, this cannot simply be amended by choosing a minimal value of l which maximises the coefficients: in the case $k_1 < k_2$, the degree bound *increases* with increasing j; this might possibly force $Q_j(x) = 0$ for low j, while allowing non-zero values for higher j. For such cases (4.3.4) could be loosened. However, in order to prove Wu's algorithm and the decoding bounds given in [24], we do not need to do that. We discuss this again in Section 4.6.

After having constructed this polynomial, all factors of the form described in the theorem should be found in order to solve the chosen instance of Problem 4.3.2. We will not discuss this process further, but an algorithm for this purpose is described by Wu in [24].

4.3.1 Analysing the bounds

The envisioned usual scenario is for Theorem 4.3.3 to be employed with known values for $\hat{\tau}$, n, k_1 and k_2, and the values of l and s should be chosen accordingly. We will analyse the bounds (4.3.4), (4.3.5) and (4.3.6) with this in mind, seeking more direct expressions for s and l based on the value of the other variables. These expressions, it turns out, will be valid only for some values of the other parameters, but for our application in coding theory, they will be sufficient.

As any satisfactory values of s and l will solve the problem equally well, we seek the values which minimises the size of the resulting equation system, with the goal of minimising consumption of time-, memory- or any other key resource consumed by a system implementing the algorithm. However, the possible values of s and l are intricately connected, and focusing on minimising one will increase the lowest possible value for the other. Without a precise description of the implementation (and possibly the architecture on which it is to run), we cannot know how they should be chosen with relation to each other in order to minimise the resource cost. Without going into these details, we can at best only optimise in a hand-waving fashion. We will do this, and derive some adequate but not necessarily optimal expressions for s and l.

The difficulty of finding a satisfying $Q(x, y)$ is roughly determined by the size of the equation system, that is, the number of linear equations times the number of variables:

$$\tfrac{1}{2}ns(s+1)(l+1)\big(s(n-\hat{\tau}) - \tfrac{1}{2}l(k_1+k_2-2)\big)$$

Focusing on s and l, this expression is – loosely speaking – dominated by the cubic in s. Therefore, if possible, it might be prudent to primarily minimise s, and thereafter choosing an l which can satisfy the bounds. That is what we will do here.

To ease notation, define first $k_\circ = k_1 + k_2 - 2$. We are going to assume $k_\circ > 0$; otherwise, we would have $k_1 = k_2 = 1$, but we will ignore this possibility: it would imply only constants as possible solutions to the rational expression interpolation, but then another obvious algorithm would be much preferred for solving the interpolation problem instead: return as solutions all constants $\tilde{y}$ which occurs as the y-coordinate in at least $n - \hat{\tau}$ of the interpolation points. Still, this case will turn out to be an anomaly when we are to assemble Wu's algorithm in Section 4.5, but for this analysis on the solution to the rational interpolation problem in general, we will assume $k_\circ > 0$.

Now the bound (4.3.4), slightly reordered becomes:

$$(l+1)\big(s(n-\hat{\tau}) - \tfrac{1}{2}lk_\circ\big) - \tfrac{1}{2}ns(s+1) > 0$$

We wish to find the minimal s for which an l exists such that the above is satisfied. If we find an expression of l as a function of s, which maximises the left-hand side of the above equation, we can then substitute that expression for l back in the above and get an inequality in s; the minimal satisfactory s in this inequality will then be the sought. Extracting the terms which are affected by l, we thus first seek to maximise the following second-degree polynomial function:

$$\begin{aligned} l(s) &= (l+1)\big(s(n-\hat{\tau}) - \tfrac{1}{2}lk_\circ\big) \\ &= -\tfrac{1}{2}k_\circ l^2 + (s(n-\hat{\tau}) - \tfrac{1}{2}k_\circ)l + s(n-\hat{\tau}) \end{aligned}$$

As $k_\circ > 0$, this has its maximal point at

$$\tilde{l}_m = \frac{s(n-\hat{\tau})}{k_\circ} - \frac{1}{2}$$

However, we are only interested in integral values of l. As a parabola is symmetric around the extremal point, we can simply choose the integer closest to it:

$$l_m = \left\lfloor \frac{s(n-\hat{\tau})}{k_\circ} \right\rfloor$$

We can of course only choose $l = l_m$ if $l_m > 0$, but we will get back to that. Further rewriting (4.3.4) and plugging this value of l into it, we need to find an s satisfying

$$\begin{aligned} (l_m+1)\big(s(n-\hat{\tau}) - \tfrac{1}{2}l_m k_\circ\big) - \tfrac{1}{2}ns(s+1) > 0 &\qquad\Longleftrightarrow \\ -\tfrac{1}{2}k_\circ l_m^2 + (s(n-\hat{\tau}) - \tfrac{1}{2}k_\circ)l_m + s(n-\hat{\tau}) - \tfrac{1}{2}ns(s+1) > 0 &\qquad\Longleftrightarrow \\ -\tfrac{1}{2}k_\circ\left(l_m - \frac{s(n-\hat{\tau})}{k_\circ} + \frac{1}{2}\right)^2 + \frac{(2s(n-\hat{\tau})-k_\circ)^2}{8k_\circ} & \\ +s(n-\hat{\tau}) - \tfrac{1}{2}ns(s+1) > 0 & \qquad (4.3.7) \end{aligned}$$

This expression becomes hard to analyse when substituting the expression for l_m, and a closed expression for the minimum of s is not easily extracted. However, as l_m was chosen as the integer closest to $\tilde{l}_m$, we have that

$$\left|l_m - \tilde{l}_m\right| = \left|l_m - \frac{s(n-\hat{\tau})}{k_\circ} + \frac{1}{2}\right| \le \frac{1}{2},$$

so (4.3.7) is definitely satisfied for all values of s satisfying

$$-\tfrac{1}{2}k_\circ\left(\frac{1}{2}\right)^2 + \frac{(2s(n-\hat{\tau})-k_\circ)^2}{8k_\circ} + s(n-\hat{\tau}) - \tfrac{1}{2}ns(s+1) > 0$$

This is simply a second degree polynomial expression in s and much easier dealt with. Using this, we are not guaranteed to find the minimum value of s, but we will most likely get a closed expression for one that is very close to. Making the straightforward simplifications, we get

$$\begin{aligned} \left(\frac{(n-\hat{\tau})^2}{k_\circ} - n\right)s^2 - \hat{\tau}s > 0 &\qquad\Longleftrightarrow \\ s > \frac{k_\circ\hat{\tau}}{(n-\hat{\tau})^2 - k_\circ n} & \qquad (4.3.8) \end{aligned}$$

as long as

$$(n-\hat{\tau})^2 > k_\circ n \tag{4.3.9}$$

Thus, a sufficient s is the lowest integral value larger than this bound, whenever this is positive:

$$s_m = \left\lfloor \frac{k_\circ \hat{\tau}}{(n-\hat{\tau})^2 - k_\circ n} + 1 \right\rfloor = \left\lfloor \frac{(n-\hat{\tau})(n-\hat{\tau}-k_\circ)}{(n-\hat{\tau})^2 - k_\circ n} \right\rfloor$$

This is always positive, as

$$\begin{aligned} (n-\hat{\tau})(n-\hat{\tau}-k_\circ) \geq (n-\hat{\tau})^2 - k_\circ n \qquad & \iff \\ k_\circ \hat{\tau} \geq 0, \qquad & \end{aligned}$$

which follows from both $k_\circ$ and $\hat{\tau}$ being positive.

As derived, the choice of $s = s_m$ assumes that we choose $l = l_m$, but this requires that $l_m \geq 1$. It turns out that this follows from the assumption (4.3.9); we wish to show

$$s_m(n-\hat{\tau}) \geq k_\circ$$

If we use the assumption (4.3.9), this definitely holds if

$$\begin{aligned} s_m n(n-\hat{\tau}) \geq (n-\hat{\tau})^2 \qquad & \iff \\ n(s_m - 1) + \hat{\tau} \geq 0, \qquad & \end{aligned}$$

which obviously holds.

Let us attempt these choices of s and l with the remaining bounds (4.3.5) and (4.3.6). First looking at the former, we need to show:

$$k_1 \leq k_2 \implies s(n-\hat{\tau}) - l(k_1 - 1) > 0$$

As we have $l_m \leq \frac{s(n-\hat{\tau})}{k_\circ}$, the right-hand side of the implication follows if

$$\begin{aligned} s_m(n-\hat{\tau}) > \frac{s_m(n-\hat{\tau})}{k_\circ}(k_1 - 1) \qquad & \iff \\ k_\circ > k_1 - 1 \qquad & \iff \\ k_2 > 1 \qquad & \end{aligned}$$

But this is implied by $k_\circ > 0$ when $k_1 \leq k_2$. Likewise, doing the exact same procedure for (4.3.6), the bound is satisfied as $k_\circ > 0 \implies (k_1 \geq k_2 \implies k_1 > 1)$. Wrapping up this analysis, we have then arrived at the following proposition:

Proposition 4.3.5 *Let $k_\circ = k_1 + k_2 - 2$ and assume $k_\circ > 0$. Assume also $(n-\hat{\tau})^2 > k_\circ n$. Choose then s and l as*

$$s = \left\lfloor \frac{(n-\hat{\tau})(n-\hat{\tau}-k_\circ)}{(n-\hat{\tau})^2 - k_\circ n} \right\rfloor$$
$$l = \left\lfloor \frac{s(n-\hat{\tau})}{k_\circ} \right\rfloor$$

We can then construct a bivariate polynomial satisfying the conditions of Theorem 4.3.3 by setting up and solving a system of $\frac{1}{2}ns(s+1)$ homogeneous linear equations in the $(l+1)\big(s(n-\hat{\tau}) - \frac{1}{2}lk_\circ\big)$ coefficients of the polynomial.

Even though these ad-hoc analyses are correct, we have not established much on the optimality of them. Perhaps the most important unanswered questions is on the requirement $(n-\hat{\tau})^2 > k_\circ n$: if that inequality is not met, will then any value of s and l satisfy the requirements (4.3.4), (4.3.5) and (4.3.6)? In order to prove Wu's algorithm and the decoding-bounds presented in [24], we do not need to answer this question, but we will briefly discuss it in Section 4.6.

4.4 Invoking rational interpolation

We will now return to coding theory and advance a step in the decoding process, using the just-defined rational interpolation problem. Section 4.2 elaborated on the results given by running the BMA on the syndromes S whenever $\epsilon > t$: If the BMA returns $A(x)$, $B(x)$ and L, then we can obtain the error-locator $\Lambda(x)$ by finding the uniquely determined polynomials $a(x)$ and $b(x)$ that satisfies

$$\Lambda(x) = A(x)a(x) + xB(x)b(x), \tag{4.4.1}$$

as well as some upper-bounds on their degrees. We obtained some more properties on these polynomials, the most important being the coprimeness of $a(x)$ and $b(x)$. That means that $\frac{b(x)}{a(x)}$ is a well-defined rational expression. What we will show in this section, is that we often know enough about this expression to be able to find it, using the interpolation algorithm presented in the previous section.

The first observation is that on all the error positions, we know the value of evaluating $\frac{b(x)}{a(x)}$ there. More precisely, let $0 \leq \ell_1 < \ell_2 < \ldots < \ell_\epsilon \leq n-1$ be the error positions. We know that for all $\tilde{x} \in \{\alpha^{-\ell_1}, \ldots, \alpha^{-\ell_\epsilon}\}$, then $\Lambda(\tilde{x}) = 0$, for that is how we defined $\Lambda(x)$. By (4.4.1), we therefore get for the same values of $\tilde{x}$, that

$$A(\tilde{x})a(\tilde{x}) + \tilde{x}B(\tilde{x})b(\tilde{x}) = 0, \tag{4.4.2}$$

This can be written as equality between two rational expressions:

$$\frac{b(\tilde{x})}{a(\tilde{x})} = -\frac{A(\tilde{x})}{\tilde{x}B(\tilde{x})} \tag{4.4.3}$$

As $a(x)$ and $b(x)$ are coprime by Proposition 4.2.4, the left-hand side is an irreducible rational expression, and as $A(x)$ and $xB(x)$ are coprime by Proposition 4.1.1 on page 41 Properties 1 and 7, the right-hand side is also an irreducible rational expression. Therefore, by (4.4.2), if $a(\tilde{x}) = 0$ then $B(\tilde{x}) = 0$ and vice versa, and (4.4.3) therefore holds true in even these cases.

So we define

$$\beta_j = -\frac{A(\alpha^{-j})}{\alpha^{-j}B(\alpha^{-j})}, \quad j = 0, \dots, n-1 \tag{4.4.4}$$

By the above, the rational expression $\frac{b(x)}{a(x)}$ goes through all points $(\alpha^{-\ell_i}, \beta_{\ell_i})$, and therefore at least ϵ of the points $(\alpha^{-0}, \beta_0), \dots, (\alpha^{-(n-1)}, \beta_{n-1})$. Most importantly; all of these points can be calculated by the receiver directly after running the BMA. We remember that some of the β_j might be ∞, and if such a j is an error position, it indicates a pole at α^{-j} in $\frac{b(x)}{a(x)}$.

We can now take the first step in using the rational interpolation problem for finding $a(x)$ and $b(x)$:

Proposition 4.4.1 *Assume $\epsilon < n-k$. Let $k_1 = \epsilon + L - (n-k)$ and $k_2 = \epsilon - L + 1$, and furthermore $\hat{\tau} = n - \epsilon$. With β_j defined as in* (4.4.4)*, the instance*

$$\big(n,\ k_1,\ k_2,\ \hat{\tau},\ [(\alpha^{-0}, \beta_0), \dots, (\alpha^{-(n-1)}, \beta_{n-1})]\big)$$

of Problem 4.3.2 has the rational expression $\frac{b(x)}{a(x)}$ among its solution.

Proof By Theorem 4.2.3 on page 50, the degrees of $a(x)$ and $b(x)$ are less than k_2 and k_1 respectively, and by Proposition 4.2.4, $a(x)$ and $b(x)$ are coprime. We showed above that the rational expression $\frac{b(x)}{a(x)}$ goes through $\epsilon = n - \hat{\tau}$ of the interpolation points. Therefore, $\frac{b(x)}{a(x)}$ is a satisfying interpolating rational expression, and must be among the problem solution. □

We can see a first important difference in the use of an interpolation problem from Wu's algorithm here and the Sudan and Guruswami-Sudan algorithms. In the latter algorithms, we search for the $n - \epsilon$ truth positions, while we here search for the ϵ error positions. This "inverts" the value of $\hat{\tau}$ in the instances, and, as we will see, causes the distinction between $\hat{\tau}$ and the error-correction bound τ.

Naturally, we now wish to use Theorem 4.3.3 in order to find a bivariate polynomial which contains the rational interpolation solution as its factors. Compared to the modelling of decoding as polynomial interpolation, we have the problem, though, that the instances described by Proposition 4.4.1 is dependent on ϵ. We would like to introduce an error-correction bound τ, and then somehow use Theorem 4.3.3 in a manner so as to find $a(x)$ and $b(x)$ whenever $\epsilon \leq \tau$. Luckily, it turns out that the bivariate polynomial we would just build for the case $\epsilon = \tau$ according to the instance from Proposition 4.4.1 can help us do just this:

Theorem 4.4.2 *Let $\tau < n-k$ and assume $\epsilon \leq \tau$. Let $k_1 = \tau + L - (n-k)$ and $k_2 = \tau - L + 1$ and furthermore $\hat{\tau} = n - \tau$. With β_j defined as in* (4.4.4)*, for the instance*

$$\big(n, k_1, k_2, \hat{\tau}, [(\alpha^{-0}, \beta_0), \ldots, (\alpha^{-(n-1)}, \beta_{n-1})]\big),$$

of Problem 4.3.2, let $Q(x,y)$ be a bivariate polynomial satisfying the conditions of Theorem 4.3.3 for some values of s and l where $l \geq s$. Then $(a(x)y - b(x)) \mid Q(x,y)$.

Proof We consider Problem 4.3.2 for instance

$$\big(n, \quad k_1 - \tau + \epsilon, \quad k_2 - \tau + \epsilon, \quad \hat{\tau} + \tau - \epsilon, \quad [(\alpha^{-0}, \beta_0), \ldots, (\alpha^{-(n-1)}, \beta_{n-1})]\big)$$

By Proposition 4.4.1, the solution to this instance contains $\frac{b(x)}{a(x)}$, and so, if $Q(x,y)$ satisfies the requirements of Theorem 4.3.3 for that instance, it follows that $(a(x)y - b(x)) \mid Q(x,y)$. We will prove that this is so, and even for the same values of s and l.

Now, directly by Theorem 4.3.3 and Proposition 4.4.1, the statement is true whenever $\epsilon = \tau$. Assume therefore $\epsilon < \tau$. As the points (a^{-i}, β_i) are independent of the value of ϵ, the first requirement of Theorem 4.3.3 is satisfied by $Q(x,y)$, so what remains is to prove that it also satisfies the second. Let D_j be the degree constraint on $Q_j(x)$ for this instance, that is:

$$D_j = s(n - (\hat{\tau} + \tau - \epsilon)) - j((k_1 - \tau + \epsilon) - (k_2 - \tau + \epsilon)) - l(k_2 - \tau + \epsilon) + (l-1),$$

and we wish to prove $\deg(Q_j(x)) \leq D_j$ for $j = 0, \ldots, l$. From the theorem's assumption and the second requirement of Theorem 4.3.3, we now have

$$\begin{aligned}
\deg(Q_j(x)) &\leq s(n - \hat{\tau}) - j(k_1 - k_2) - lk_2 + (l-1) \\
&= s(n - (\hat{\tau} + \tau - \epsilon)) + s(\tau - \epsilon) - j((k_1 - \tau + \epsilon) - (k_2 - \tau + \epsilon)) \\
&\qquad - l(k_2 - \tau + \epsilon) - l(\tau - \epsilon) + (l-1) \\
&= D_j - (l-s)(\tau - \epsilon) \\
&\leq D_j,
\end{aligned}$$

where the last inequality is due to $l \geq s$ and $\tau > \epsilon$. This finishes the proof. □

That is, if we can construct this one bivariate polynomial $Q(x,y)$, then as long as the number of errors satisfies $\epsilon \leq \tau < n-k$, we can find $a(x)$ and $b(x)$ by factoring $Q(x,y)$. From $a(x)$ and $b(x)$, we immediately get $\Lambda(x)$ using (4.4.1), and with $\Lambda(x)$, we can find the error positions and then the values. For the most part, we are then done as long as we can construct such a $Q(x,y)$.

Using the analysis conducted in Section 4.3.1, we can even immediately describe this construction. To apply this, however, we must limit τ. After the proof of the following corollary, we will shortly discuss these limitations. Now for the result:

Corollary 4.4.3 *Let $\frac{1}{2}(n-k+1) < \tau < \min(n-k,\ n-\sqrt{n(k-1)})$. Choose now s and l as*

$$\begin{aligned} s &= \left\lfloor \frac{\tau(n-k+1-\tau)}{\tau^2 - n(2\tau-(n-k+1))} \right\rfloor \\ l &= \left\lfloor \frac{s\tau}{2\tau-(n-k+1)} \right\rfloor \end{aligned}$$

We can then construct a bivariate polynomial satisfying the conditions of Theorem 4.4.2 by setting up and solving a system of $\frac{1}{2}ns(s+1)$ homogeneous linear equations in the $(l+1)\big(\frac{1}{2}l(n-k+1)-(l-s)\tau\big)$ coefficients of the polynomial.

Proof The choices of s and l and the remaining result follows from Proposition 4.3.5, substituting the values for k_1, k_2 and $\hat{\tau}$ given in Theorem 4.4.2. We only need to show the assumptions for Proposition 4.3.5 as well as the one from Theorem 4.4.2.

- $k_\circ > 0$: Expanding the variables, we have

$$\begin{aligned} k_\circ &= k_1 + k_2 - 2 \\ &= (\tau + L - (n-k)) + (\tau - L + 1) - 2 \\ &\geq 2\tau - (n-k+1) \end{aligned}$$

  which, by the corollary's assumption, is greater than zero.

- $(n-\hat{\tau})^2 > k_\circ n$: Substituting for the parameter's values, we get

$$\begin{aligned} (n-\hat{\tau})^2 > k_\circ n &= (k_1+k_2-2)n && \iff \\ \tau^2 &> \big((\tau+L-(n-k)) + (\tau-L+1)-2\big)n && \iff \\ 0 &< \tau^2 - 2\tau n + (n-k+1)n && \implies \\ \tau &< n - \sqrt{n(k-1)}, \end{aligned}$$

  which was an assumption to the corollary.

- $l \geq s$: By the corollary's assumption, the denominator in the expression for l is always positive. By that expression, we then only need to show

$$\begin{aligned} &\tau \geq 2\tau - (n-k+1) && \iff \\ &\tau \leq n-k+1, \end{aligned}$$

  which follows from $\tau < n - \sqrt{n(k-1)} < n-k+1$. □

As we are going to use these results in the decoding algorithm only when $\epsilon > t$, notice that the lower bound $2\tau > n-k+1$ can only be unsatisfied when $\tau = \frac{1}{2}(n-k+1) = t+1$; that is, when $n-k$ is odd and τ only one greater than the minimum-distance decoding bound t. As we will see, this special case leads to a separate, degenerate version of Wu's decoding algorithm.

Notice also that the bound $\tau < n-k$ is almost always implied by the bound $\tau < n-\sqrt{n(k-1)}$. This last bound is the Johnson bound, which we also met with the Guruswami-Sudan algorithm. It is interesting how this bound falls out of the equations in a seemingly coincidental manner. Also, it is not at all clear whether it could be exceeded, had we performed the analysis in Section 4.3.1 in a different manner. Even though *any* polynomial-time list decoder must be bound by $\tau < n-k$, as we discussed in Section 1.2.2, it is often not stated, when the decoder is also bound by the Johnson bound, as the latter implies the former for all practical values of n and k.

We have now modelled the search for $a(x)$ and $b(x)$ as a rational interpolation problem and showed how to solve it. The final step is to collect all the results together in a complete algorithm, describing the complete process.

4.5 Putting it all together

We will now collect all the bits and pieces found in this chapter, and describe and prove the assembled Wu's algorithm. We will still refer to $A(x)$, $B(x)$ and L as the result of running the BMA on S, and refer to $a(x)$ and $b(x)$ as the uniquely determined polynomials satisfying the conditions of Theorem 4.2.3 on page 50.

Because of the special case of $\tau = \frac{1}{2}(n-k+1)$ identified by the proof of Corollary 4.4.3, we are forced to split the algorithm in two: the general one, where we assume $\tau > \frac{1}{2}(n-k+1)$, and then, in Section 4.5.1, the lesser one, where $\tau = \frac{1}{2}(n-k+1)$. The general one presented in this Section is naturally the main result, while the other algorithm is mostly for completeness.

We give the general Wu's algorithm immediately; it is markedly more complicated than any of the previous decoding algorithms:

Algorithm 4.5.1 (Wu's list decoding)

1. Choose a desired value for the error correction capacity τ with

   $$\tfrac{1}{2}(n-k+1) < \tau < \min(n-k,\ n-\sqrt{n(k-1)})$$

   Set s and l according to Corollary 4.4.3.

2. Run the BMA on S, which returns $A(x), B(x)$ and L.

3. If $L = \deg(A(x))$ and $A(x)$ has L distinct non-zero roots, then we guess that $A(x) = \Lambda(x)$. Find the error positions from this, and thereafter the error values using Forney's algorithm. If the result is a valid codeword, return the corresponding information word and break.
4. If not $n - k - \tau < L \leq \tau$, declare a decoding failure and break.
5. Construct a $Q(x, y)$ satisfying the conditions of Theorem 4.4.2.
6. Find all factors of $Q(x, y)$ of the form $(a^\star(x)y - b^\star(x))$ where $\deg(a^\star(x)) < k_2$ and $\deg(b^\star(x)) < k_1$, where k_1 and k_2 are defined as in Theorem 4.4.2.
7. For each found $(a^\star(x), b^\star(x))$ construct a possible error-locator as
$$\Lambda^\star(x) = A(x)a^\star(x) + xB(x)b^\star(x)$$
8. For each such $\Lambda^\star(x)$, if it has $\deg(\Lambda^\star(x))$ distinct non-zero roots, then find the error positions and thereafter the error values using Forney's algorithm, and construct a possible codeword $c^\star$.
9. Remove all of these $c^\star$ which are not valid codewords. Among the rest, select the ones with minimum distance from r. From these, retrieve the corresponding information words and return as equally likely solutions.

■

That the algorithm functions correctly is captured by the following theorem:

Theorem 4.5.2 *Wu's algorithm is a τ-bounded-distance decoder.*

Proof Assume that $\epsilon \leq \tau$ and no codewords have distance less than ϵ from r. We must then prove that the algorithm returns a list where the sent information word w is on.

In case $\epsilon \leq n-k-L$, Proposition 4.2.6 on page 51 tells us that $(\Lambda(x), \epsilon) = (A(x), L)$. Therefore, the tests in Step 3 will pass, and the information word will be returned on a singleton list.

Otherwise, we have

$$\begin{aligned} &\epsilon > n - k - L && \Longrightarrow \\ &L > n - k - \epsilon \geq n - k - \tau \end{aligned}$$

And by Proposition 4.2.5 on page 51, we must also have $L \leq \epsilon \leq \tau$. Therefore, the algorithm will not break in Step 4.

By Theorem 4.4.2, the bivariate polynomial constructed in Step 5 will contain the factor $(a(x)y - b(x))$, so it will be among the found $(a^\star(x), b^\star(x))$ in Step 6. By Theorem 4.2.3 on page 50, the equation

$$\Lambda(x) = A(x)a(x) + xB(x)b(x)$$

is satisfied. Therefore, $\Lambda(x)$ will be among the found $\Lambda^\star(x)$ from step 7. As a true error-locator, it has $\deg(\Lambda(x))$ distinct roots, so the sent codeword c will be among the found $c^\star$ in Step 8.

As we assumed that no codewords have distance less than ϵ from r, the information word w will be among the returned words. □

An alternative formulation of a τ-bounded-distance list decoder is that it should return *all* codewords within distance τ of r. If we slightly change Step 9 to include all found codewords within distance τ, Wu's algorithm can be seen to conform to this formulation as well.

For an algorithm to be usable, we also need to be sure that it can be realised. Most of the steps include only known procedures, but Step 5 and Step 6 deserve more attention.

According to Corollary 4.4.3, we can construct a satisfying bivariate $Q(x, y)$ polynomial in Step 5 by solving a homogeneous system of linear equations, as long as the conditions on τ are met; but those are equivalent to our bounds when choosing it in Step 1. Also, in Step 1, the algorithm chooses s and l as required. Therefore, the homogeneous system of linear equations can be set up and solved with any general method, such as Gaussian Elimination.

Step 6 assumes an algorithm for finding factors of the specific form $(a^\star(x)y-b^\star(x))$ in a bivariate polynomial. As already mentioned in Section 4.3, it is outside the scope of this thesis do describe such an algorithm. Wu himself gives a fast one, utilising the known bounds on the degrees of the polynomials $(a^\star(x), b^\star(x))$ of interest [24].

Note that Step 4 is not essential; it merely saves the time of constructing the $Q(x, y)$ polynomial in certain cases where we know that no codewords can be within distance τ of r.

We have then finally arrived at what we set out for: presenting and proving Wu's list-decoding algorithm. In Section 4.6, we will discuss various aspects of the derivation and possible places for improvements, but first, we deal with the outlying case $\tau = \frac{1}{2}(n - k + 1)$ in the following section.

4.5.1 The curious case of $\tau = \frac{1}{2}(n - k + 1)$

As we saw in Section 4.4, whenever $\tau = \frac{1}{2}(n-k+1)$, the general approach of solving the rational interpolation problem, provided by Section 4.3 and its subsequent analysis in Section 4.3.1, fails. This calls for a degenerate version of the general Wu's algorithm.

Having $\tau = \frac{1}{2}(n - k + 1)$ is only possible whenever $n - k$ is odd. In this case, we have

$$t = \left\lfloor \frac{n-k}{2} \right\rfloor = \tfrac{1}{2}(n-k-1) = \tau - 1$$

Therefore, decoding to $\frac{1}{2}(n-k+1)$ is precisely one step further than usual minimum-distance decoding. As for the general Wu's algorithm, we will first run the BMA on the syndromes, so whenever $\epsilon \leq t$, the results of Section 2.5 guarantees that we decode correctly. Therefore, what remains is only the case $\epsilon = \tau = t+1$.

We will now present the algorithm, and thereafter prove that it works correctly.

Algorithm 4.5.3 (Wu's One-Step-Ahead decoding)

1. Set $\tau = \frac{1}{2}(n-k+1)$
2. Run the BMA on S, which returns $A(x), B(x)$ and L.
3. If $L = \deg(A(x))$ and $A(x)$ has L distinct non-zero roots, then we guess that $A(x) = \Lambda(x)$. Find the error positions from this, and thereafter the error values using Forney's algorithm. If the result is a valid codeword, return the corresponding information word and break.
4. If $L \neq \tau$, declare a decoding failure and break.
5. For all $j = 0, \ldots, n-1$, define

   $$\beta_j = -\frac{A(\alpha^{-j})}{\alpha^{-j}B(\alpha^{-j})}, \quad j = 0, \ldots, n-1$$

   Divide the indices $j = 0, \ldots, n-1$ into sets such that two indices j_1 and j_2 are in the same set if and only if $\beta_{j_1} = \beta_{j_2}$.
6. For each of these sets $G^\star$, if $|G^\star| = \tau$, let its elements different be denoted $\ell_1^\star, \ldots, \ell_\tau^\star$. These are error positions corresponding to one possible decoding. Construct a corresponding error-locator as

   $$\Lambda^\star(x) = \prod_{i=1}^{\tau}(1 - \alpha^{\ell_i^\star} x),$$

7. For each such $\Lambda^\star(x)$, find the error values using Forney's algorithm, and perform error correction into a possible codeword $c^\star$.
8. Remove all of these $c^\star$ which are not valid codewords. From the rest, retrieve the corresponding information words and return as equally likely solutions.

■

Even though the grouping of the indexes done in step 5 of the algorithm looks alien, it corresponds exactly to rational interpolation for the degenerate case where the rational expressions we search for are actually constants. This is also reflected in the proof of the correctness of the algorithm:

Theorem 4.5.4 *Wu's One-Step-Ahead algorithm is a $\frac{1}{2}(n-k+1)$-bounded-distance decoder.*

Proof Assume that $\epsilon \le \tau = \frac{1}{2}(n-k+1)$ and that no codewords have distance less than ϵ from r. We must then prove that the algorithm returns a list of information words with the sent word w on it.

As for the proof of the general Wu's algorithm, if the $\epsilon \le t$, the algorithm will return w on a singleton list in step 3. If we reach step 4, we then know that $\epsilon = \tau$. By Proposition 4.2.5 on page 51, we have $L \le \tau$. On the other hand, from Theorem 4.2.3 on page 50, we know there exists $a(x)$ and $b(x)$ satisfying the degree bounds

$$\begin{aligned} \deg(a(x)) &\le \tau - L \\ \deg(b(x)) &\le \tau + L - (n-k+1) \end{aligned}$$

If $b(x) = 0$, Proposition 4.2.4 on page 51 gives $a(x) = 1$, but that contradicts $\Lambda(x) \ne A(x)$. Therefore, the degree of $b(x)$ must be non-negative, which, with the degree-bound above, implies $L \ge \tau$. Therefore, we have $L = \tau$, and the algorithm will not break in Step 4.

With $L = \tau$ used in the degree bounds, we see that $a(x)$ and $b(x)$ must both be constant expressions, so $\frac{b(x)}{a(x)}$ is also constant. But then by Proposition 4.4.1 on page 62, those β_j corresponding to error-locations, of which there are $\epsilon = \tau$, will all equal this constant.

Therefore, one of the sets assembled in Step 5 will consist of the error-locations. Therefore, $\Lambda(x)$ will be among the found $\Lambda^\star(x)$ in Step 6; c will be among the codewords in Step 7; and w will be among the returned information words. □

Wu points out that this algorithm has the same asymptotic running time as the Berlekamp-Massey decoder, but can decode one more error [24]. So even though it is not as powerful as the general Wu's list decoder, it has its merits as being a fast decoder for going beyond the minimum distance.

In [24], Wu presents this algorithm as an appetiser before giving the general list decoder. However, he seems to neglect that this algorithm is actually necessary whenever $\tau = \frac{1}{2}(n-k+1)$, even though his (and our) closed expressions for l and s for the general algorithm will include a division by zero in this case. Of course, whenever $\tau = \frac{1}{2}(n-k+1)$, one would always employ this much faster algorithm, rather than a full-fledged list-decoding algorithm, even if the general Wu's algorithm could support it.

4.6 Discussion

We will now discuss various aspects of Wu's algorithm, the derivation we have just given, as well as areas which might be interesting to look into in future work.

Down the rabbit-hole

The analyses, remarks and observations on Wu's algorithm given in this thesis are by no means complete, and there is still a lot left to examine.

To judge the efficiency of Wu's algorithm with respect to earlier algorithms – especially the Guruswami-Sudan and its derivatives, which have the same decoding bound – it is necessary to give a running-time analysis of the algorithm. It has been outside the scope of this project to deal with running-time in any detailed manner. Indeed, for these advanced algorithms, it is not at all an easy task to compare the running time. Wu himself gives an analysis in the case of choosing τ maximally high, and arrives at an asymptotic running time; however, this is not in any precise sense compared to the running time of the Guruswami-Sudan algorithm or any of its derivatives. He does compare the needed values of the multiplicity s and the list size l required when choosing τ maximally, and concludes that the values required for Wu's algorithm are universally lower – and in the case of s, much lower – than those of the Guruswami-Sudan; this certainly bodes well, but is not itself conclusive. The parameters of the two algorithms impact the total complexity quite differently. It would be very interesting to make a precise analysis of the running time and compare it with the Guruswami-Sudan algorithms and its derivatives; also for lower values of τ. It would also be interesting to compare the running time with that of some minimum-distance decoders, and get a measure for the added running time per extra error that is to be corrected.

One obvious advantage that Wu's algorithm has over the Guruswami-Sudan, is that whenever $\epsilon \leq t$, it is as fast as the Berlekamp-Massey decoder, and that in cases where $\epsilon > t$, the preliminary run of the BMA is not "wasted". This can be utilised in several ways. One could e.g. imagine a two-processor decoding computer: the first processor decodes the received words as they arrive, most of which have not more than t errors. The few which do are sent to the second processor along with the result of running the BMA on its syndromes. The second processor is then in charge of the remaining part of Wu's algorithm for these few received words. In cases of heavy load, if the second processor cannot keep up, the worst impact is that the decoding computer degenerates to a minimum-distance decoder. It is also very cheap to save the received word along with the result of the BMA for later, when more computing time is available. This way, list decoding can come almost for "free". Naturally, a similar setup could be done with the Berlekamp-Massey decoding algorithm in parallel with the Guruswami-Sudan, but the time spent in the first would be completely wasted when having to run the second.

The value of the list size l in both the Guruswami-Sudan and Wu's algorithm determine the maximal number of codewords that could be returned. If Wu's algorithm requires a lower value of l for a given decoding-bound, a detailed analysis might arrive at new, tighter upper-bounds for the number of codewords in a given distance of any word. Before doing this, it would be interesting to redo the analysis in 4.3.1

with the goal of minimising l for a given τ, instead of minimising s as we did.

Through the looking glass

It is of course of utmost interest to improve on Wu's algorithm in any way: speed improvements, loosening the upper bound etc. We have given some thoughts on possible areas of future work.

The most obvious place to look, is to examine the improvements on the Guruswami-Sudan algorithm suggested in the last decade, e.g. [1,2,18]. Some of these – especially the ones concerned only with the interpolation part of the Guruswami-Sudan – might be applicable to Wu's algorithm as well.

The analysis done in 4.3.1 was ad-hoc, in the sense that it did not prove that the assumptions reached in Proposition 4.3.5 on page 61 were necessary; that is, whenever the assumptions are not met, is it then impossible to set up a system of linear equations to find a satisfying bivariate polynomial $Q(x, y)$? In particular, if something could be done for the requirement $(n - \hat{\tau})^2 > k_\circ n$, we might be able to loosen the upper bound on τ when invoking the rational interpolation solution in Section 4.4.

We remarked in Section 4.3, that when finding the bound (4.3.4), for describing when we can find $Q(x, y)$ by setting up and solving a system of linear equations, the bound will be tighter than necessary whenever some of the degree-bounds in the second requirement of Theorem 4.3.3 are negative. Especially in the case where $k_1 < k_2$, we cannot employ the same trick as for the Sudan and Guruswami-Sudan algorithms, and choose a minimum l that maximises the coefficients; in this case, the degree-bounds will *increase* with increasing j. In this case, we might be able to improve on (4.3.4) and the following analysis in Section 4.3.1. However, on the face of it, these speculations have little interest in the application of list-decoding: whenever $\Lambda(x) \neq A(x)$, we must have $\epsilon > \frac{1}{2}(n - k)$, so Proposition 4.2.6 on page 51 gives

$$\epsilon + L > n - k \iff 2L > n - k,$$

but when we are invoking rational interpolation in Section 4.4, the expressions for k_1 and k_2 then gives

$$\begin{aligned} 2L &> n - k && \iff \\ \tau + L - (n - k) &\geq \tau - L + 1 && \iff \\ k_1 &\geq k_2 \end{aligned}$$

So whenever we reach the rational interpolation, we know that $k_1 \geq k_2$.

It should be noted, however, that there seems to be nothing preventing us from searching for the rational expression $\frac{a(x)}{b(x)}$ instead of $\frac{b(x)}{a(x)}$ in Section 4.4 (as Proposition 4.2.4 on page 51 ensures us that $b(x) \neq 0$ whenever $\Lambda(x) \neq A(x)$). In that case, we would *always* have the case $k_1 < k_2$ when reaching the rational interpolation, and the above could then be looked into.

One part of Wu's algorithm that seems counter-intuitive is that s and l are chosen *before* looking at the received word. After having thoroughly examined the received word by running the BMA on the syndromes, we should be able to tell a lot more about the possibilities of nearby codewords; the Propositions 4.2.5 and 4.2.6 on page 51 attest to that. Why is it then, that we cannot improve on the value of these key parameters s and l? It seems like a coincidence from the requirements of Theorem 4.3.3 and its analysis afterwards, that it is the sum of k_1 and k_2 which is used in the expressions of s and l, instead of the values individually; the latter could have made s and l depend on L. It could be interesting to make a deeper analysis to reveal if this is something general, and that we can gain nothing from setting s and l after running the BMA, or if it is only a by-product of the path taken to reach Wu's algorithm.

Related to this, Proposition 4.2.5 on page 51 ensures that, as soon as we have run the BMA, we actually do not need to look for codewords with fewer errors than L. However, in the invocation of rational interpolation in Section 4.4, the final $Q(x, y)$ is so general that it can solve the problem for any number of errors at most τ. This does not seem tight; however, it is not clear if anything could be tightened up to avoid it.

Adventures in Wonderland

Feng and Tzeng developed in [7] the Fundamental Iterative Algorithm; a general matrix-manipulation algorithm, which has the interesting property that it equals the Berlekamp-Massey algorithm when applied to the decoding setting and the Hankel form of the S matrix in (2.3.1) on page 17. It could be interesting to derive the results of Sections 4.1 and 4.2 from this angle.

Related to this, it is well-known that the BMA is in some sense equivalent to the Extended Euclidean algorithm for polynomials (EA); see e.g. [12]. Therefore, it should be possible to have Wu's algorithm backed by the EA instead of the BMA. As the background theory for the EA is algebraically much simpler and more well-studied than for the BMA, it might prove to be interesting from a theoretical point of view and reveal properties of the algorithm and the codes which lie more hidden in Wu's algorithm as it is.

Chapter 5

Conclusion

Recently, Wu presented in [24] a new Reed-Solomon list-decoding algorithm, which combines the classic Berlekamp-Massey algorithm with ideas from the Guruswami-Sudan algorithm in a new and ingenious fashion. It has the same decoding-bound as the Guruswami-Sudan, while promising lower running time. Most importantly, this new approach to list-decoding might shed more light on the properties of the Reed-Solomon codes. However, the impact of this discovery has of yet been very limited; this might be attributable to the compactness of its sole presentation in one, condensed article.

In this thesis, we have given an new presentation of Wu's algorithm, as well as the entire evolution of background theory leading up to it. The hope is that such a full-length, consistent presentation will help in a deeper theoretical analysis of this new angle to list-decoding.

First, we showed well-known results in linear recursions leading to the Berlekamp-Massey decoding algorithm, and then the Guruswami-Sudan algorithm with its solution to the polynomial interpolation problem.

Using the same notation, tailored for this thesis, we then proceeded with the improvements to the known theory needed to show the results of Wu and for proving the correctness of Wu's algorithm. In Wu's original paper [24], all results leading to the algorithm were proved only in the setting of the decoding problem. Instead, we have sought to detach as many results as possible from this application.

This first led to new results on linear recursions in general and on the Berlekamp-Massey algorithm for solving these. We introduced the concept of completion-pairs, for which we proved several important properties.

The key part of the Guruswami-Sudan algorithm, for solving the polynomial interpolation, was generalised to handle rational expressions in [24]. We gave a new formulation of this generalisation, independent of coding theory, and we also gave a new analysis of the bounds on its parameters, leading to closed expressions for the auxiliary parameters: the multiplicity and the list size.

With this theory in hand, we have attempted to minimise the proofs needed for showing the results for the decoding problem and arriving at Wu's algorithm. The aim has been to reveal, as clearly as possible, the suboptimal parts of the derivation; where limitations to e.g. the decoding-bound arise; and which relations this new approach has to previous work. We have discussed a number of these areas, which we believe could be interesting for future work.

Appendix A

List of symbols

This is a short description of most of the "globally defined" symbols and variables used throughout the thesis; in particular, they contain the ones that we might use far from their original definition. It continues on the next page.

k	The length of the information word. Known by both sender and receiver in advance.
n	$= q - 1 > k$. The length of the codeword. Known by both sender and receiver in advance.
d	$= n - k + 1$. The minimum distance of Reed-Solomon codes.
w	$= (w_0, \ldots, w_{k-1})$. The intended information word from the sender. Unknown to the receiver. Also $w(x) = \sum_{i=0}^{k-1} w_i x^i$
c	$= (c_0, \ldots, c_{n-1})$. The sent codeword. Unknown to the receiver.
r	$= (r_0, \ldots, r_{n-1}) = c + e$. The received codeword. Also $r(x) = \sum_{i=0}^{n-1} r_i x^i$
e	$= (e_0, \ldots, e_{n-1}) = r - c$. The error vector. Unknown to the receiver. Also $e(x) = \sum_{i=0}^{n-1} e_i x^i$
ϵ	The number of errors, i.e. the number of non-zero positions in e. Unknown to the receiver.
t	$= \lfloor \frac{1}{2}(n - k) \rfloor$. The decoding bound of a minimum-distance decoder.
$\mathbb{F}_q$	The finite field with q elements on which our code works.

α	A primitive element in the field $\mathbb{F}_q$, which we use for our codes.
S	$= (S_1, \dots, S_{n-k})$. The syndromes of the received word r. Directly calculable by the receiver.
E	$= (E_1, \dots, E_n)$. The error-spectrum of the error e. Unknown to the receiver. We have $S_i = E_i$ for the first $n-k$ elements. Also $E(x) = \sum_{i=0}^{n-1} E_{i+1}x^i$.
ℓ_i	The ϵ error-positions. That is, the indexes j for which $e_j \neq 0$. Unknown to the receiver.
$\Lambda(x)$	$= \prod_{i=1}^{\epsilon}(1 - x\alpha^{-\ell_i})$. The error-locator. Unknown to the receiver.
$A(x)$, $B(x)$, L	The result of running the BMA; $(A(x), L)$ is a shortest linear recursion producing the input, and $B(x)$ is the helper-polynomial. In Section 4.2 and onwards, the result from running the BMA on S.
$A^{(i)}(x)$, $B^{(i)}(x)$, $L^{(i)}$	The i'th iteration's sub-results when running the BMA. $(A^{(i)}(x), L^{(i)})$ is a shortest linear recursion producing the first i elements of the input.
$\delta^{(i)}$	The i'th iteration's discrepancy when running the BMA; that is, the difference between the i'th element of the input sequence and the i'th production of the linear recursion $(A^{(i-1)}(x), L^{(i-1)})$.
$\flat^{(i)}$	$= [\delta^{(i)} \neq 0 \ \wedge \ L \neq L^{(i-1)}]$
$a(x), b(x)$	In Section 4.2 and onwards, the companion-polynomials which in polynomial combination with $A(x)$ and $B(x)$ give $\Lambda(x)$; these are unknown to the receiver. See Theorem 4.2.3 on page 50.
τ	The error-correction bound for list-decoders. Usually greater than t. Also the number of points allowed to be incorrect in an interpolating polynomial in Chapter 3.
$\hat{\tau}$	The number of points allowed to be incorrect in an interpolating rational expression. Used in Section 4.3.
l	The list size parameter in our interpolation algorithms.
s	The multiplicity parameter of our interpolation algorithms.
k_1, k_2	In the rational interpolation problem, the upper-bound on the degree of the numerator, respectively denominator, of an interpolating polynomial.

Bibliography

[1] M. Alekhnovich, *Linear diophantine equations over polynomials and soft decoding of Reed-Solomon codes*, IEEE Transactions on Information Theory, 51 (2005).

[2] P. Beelen and K. Brander, *Key-equations for list decoding of Reed-Solomon codes and how to solve them*, Submitted to Journal of Symbolic Computation, (2009).

[3] E. R. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, 1968.

[4] R. E. Blahut, *Algebraic Codes on Lines, Planes, And Curves*, Cambridge University Press, April 2008.

[5] Y. Cassuto and J. Bruck, *A Combinatorial Bound on the List Size*, tech. rep., California Institute of Technology, August 2004.

[6] R. Chien, *Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes*, IEEE Transactions on Information Theory, 10 (1964), pp. 357–363.

[7] G.-L. Feng and K. Tzeng, *A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes*, IEEE Transactions on Information Theory, 37 (1991), pp. 1274–1287.

[8] V. Guruswami and A. Rudra, *Limits to list decoding Reed-Solomon codes*, in STOC '05, ACM, 2005, pp. 602–609.

[9] ——, *Explicit Capacity-Achieving List-Decodable Codes*, in Proceedings of STOC '06, ACM, 2006, pp. 1–10.

[10] V. Guruswami and M. Sudan, *Improved decoding of Reed-Solomon codes and algebraic geometry codes*, IEEE Transactions on Information Theory, 45 (1999), pp. 1757–1767.

[11] ——, *Extensions to the Johnson bound*, tech. rep., MIT Laboratory for Computer Science, 2001.

[12] A. E. Heydtmann and J. M. Jensen, *On the Equivalence of the Berlekamp-Massey and the Euclidean Algorithms for Decoding*, IEEE Transactions on Information Theory, 46 (2000), pp. 2614–2624.

[13] S. M. Johnson, *A new upper bound for error-correcting codes*, IEEE Transactions on Information Theory, 46 (1962), pp. 203–207.

[14] J. Justesen and T. Høholdt, *A Course in Error-Correcting Codes*, European Mathematical Society, 2004.

[15] D. E. Knuth, *Two notes on notation*, American Mathematical Monthly, 99 (1992), pp. 403–422.

[16] F. J. MacWilliams and N. J. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing, 1977.

[17] J. Massey, *Shift-register synthesis and BCH decoding*, IEEE Transactions on Information Theory, 15 (1969), pp. 122–127.

[18] V. N. Muralidhara and S. Sen, *Improvements on the Johnson bound for Reed-Solomon codes*, Discrete Applied Mathematics, 157 (2009), pp. 812–818.

[19] C. E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal, 27 (1948), pp. 379–423, 623–656.

[20] G. Solomon and I. S. Reed, *Polynomial Codes Over Certain Finite Fields*, Journal of the Society for Industrial and Applied Mathematics, 8 (1960), pp. 300–304.

[21] H. Stichtenoth, *Algebraic Function Fields and Codes*, Springer, 2nd ed., 2009.

[22] M. Sudan, *Decoding of Reed Solomon codes beyond the error-correction bound*, Journal of Complexity, 13 (1997), pp. 180–193.

[23] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, *A Method for Solving Key Equation for Decoding Goppa Codes*, Information and Control, 27 (1975), pp. 87–99.

[24] Y. Wu, *New List Decoding Algorithms for Reed-Solomon and BCH Codes*, IEEE Transactions on Information Theory, (2008).