Title: Design of An Efficient List Decoding System for Reed-Solomon and Algebraic-Geometric Codes

Table of Contents:

Abstract

This thesis presents an efficient list decoding system for Reed-Solomon codes and algebraic-geometric codes. Reed-Solomon codes are non-binary block codes, which are widely used in communication and storage systems. However, the availability of the codes and their performance are limited by the code length which can not exceed the size of finite field. Compared with Reed-Solomon codes which are defined in the same finite field, algebraic-geometric codes are usually longer and there are more available codes. Their longer code length results larger designed minimum distance which enables the code to correct more errors in a code word frame. Therefore, algebraic-geometric codes tend to be a suitable replacement for Reed-Solomon codes in future applications.

The list decoding system can correct errors beyond the half distance boundary which is the capability bottleneck for the conventional unique decoding algorithm. This project's research produces three main contributions to the list decoding system with respect to its decoding efficiency and wider application to algebraic-geometric codes. First, a general complexity reduction scheme for the complexity dominant interpolation process is proposed. The scheme can be applied list decoding of Reed-Solomon and algebraic-geometric codes, as well as hard and soft decision decoding systems. Second, the list decoding process of Hermitian code has been engineered with a clear mathematical framework. The first simulation results for list decoding of Hermitian codes are presented, showing significant coding gains can be achieved over the unique decoding algorithm. For improving the efficiency of list decoding of Hermitian codes, a supported algorithm for calculating the key parameters (the corresponding coefficients between a Hermitian curve's pole basis monomials and zero basis functions) is proposed. Third, the first soft-decision list decoder for Hermitian codes has been developed, in which a priori process that obtains the received information's reliability values and converts them into interpolation multiplicity values is introduced. During this conversion, a practical stopping rule based on the designed length of output list is proposed. The obtained simulation results show that further improvement can be achieved with over the hard-decision decoding scheme, but with only small increase in decoding complexity.

Chapter 1

Introduction

1.1 Introduction

Algebraic-geometric codes were first introduced by Goppa [1], showing error-correction codes can be constructed from algebraic curves. Among them, Reed-Solomon codes [2] which were introduced in the 1960s are in fact the simplest algebraic-geometric codes and are constructed from a straight line. Today, Reed-Solomon codes are widely used in both communications and storage systems. Compared with Reed-Solomon codes, general algebraic-geometric codes that are constructed from the same finite field have longer code lengths, resulting in a larger minimum distance of the code and hence more errors can be corrected in a code word frame. Therefore, algebraic-geometry codes are suitable to replace Reed-Solomon codes in future advanced applications.

To apply algebraic-geometric and Reed-Solomon codes, developing a decoding algorithm with good error-correction capability and low decoding complexity is important. For algebraic-geometric and Reed-Solomon codes, the most conventional and efficient decoding algorithms are called unique decoding algorithms. Specifically, for Reed-Solomon codes, the best unique decoding algorithm is the Berlekamp-Massey algorithm [3, 4], while for algebraic-geometric codes, the best unique decoding algorithm is the Sakata algorithm [5] with Feng and Rao's majority voting [6]. The unique decoding algorithm determines syndromes from the received word. Then based on the syndromes, error locations and magnitudes are calculated in order to recover the correct transmitted code word. The main limitation of the unique decoding algorithms is their error-correction which cannot exceed the half distance bound $\left\lfloor \frac{d-1}{2} \right\rfloor$, where d indicates the minimum Hamming distance of the code.

To achieve better error-correction capability for algebraic-geometric and Reed-Solomon codes, the list decoding algorithm is an alternative choice. An improved list decoding scheme which can correct errors beyond the half distance bound for both algebraic-geometric and Reed-Solomon codes was introduced by Guruswami and Sudan [7] in 1999. This decoding algorithm can provide better performance than the conventional unique decoding algorithms. Based on Guruswami-Sudan's list decoding scheme, further improvements can be achieved by the soft-decision list decoding algorithm, which was introduced by Koetter and Vardy [8] in 2003 for Reed-Solomon codes.

1.2 Motivation and Challenges

Reed-Solomon codes are widely applied by the communication and storage industry, such as deep-space satellite communications, high-speed modems, compact disc (CD), hard drive, digital versatile disc (DVD) etc. As they are constructed from the affine points on a straight line, the size of Reed-Solomon codes cannot exceed the size of the finite field over which it is defined. Therefore, for more advanced applications, long codes with better error-correction capability are required. To achieve this, Reed-Solomon codes must be defined over a larger finite field. But as a consequence, the decoding complexity will be increased exponentially. Another solution is to use more advanced coding schemes – algebraic-geometric codes. Compared with Reed-Solomon codes defined over the same finite field, algebraic-geometric codes are longer as they are constructed from the affine points of an algebraic curve rather than a straight line, resulting in greater errors can be corrected in a code word frame. Therefore, more powerful algebraic-geometric codes can be constructed from a moderate size of finite field.

Before 1999, the best known decoding algorithms for Reed-Solomon and algebraic-geometric codes could only correct errors up to the half distance bound, limiting these codes' performance over deeply corrupted scenarios. In 1999, Guruswami and Sudan's list decoding scheme [7] exceeded this bound for both Reed-Solomon and algebraic-geometric codes. The general idea of Guruswami-Sudan's list decoding scheme is to reconstruct a list of most likely transmitted code words based on a given received word. This code word reconstruction is performed by two processes: interpolation and factorisation. Building on the work of Guruswami and Sudan, Høholdt and Nielsen [9] presented a mathematical framework for the list decoding of one of the best performing algebraic-geometric codes – Hermitian codes. Koetter and Vardy [8] showed that further list decoding improvements for Reed-Solomon codes can be achieved by a soft-decision scheme. It is also realised that this performance improvement only introduces a small increase in complexity compared with Guruswami-Sudan's hard-decision list decoding scheme.

Although the list decoding algorithm can produce better performance, it is at the expense of higher decoding complexity compared to the conventional unique decoding algorithms. So far, few papers in the literature have addressed this problem as it is still a new decoding algorithm which is not well known by many researchers. The only performance evaluation on soft-decision and hard-decision list decoding of Reed-Solomon codes appeared in Koetter and Vardy's paper [8], but there is still a lack of analysis on how decoding complexity changes with regards to the critical decoding parameter – interpolation multiplicity. For list decoding of algebraic-geometric codes, a soft-decision scheme is yet to be developed and there is no performance evaluation with regards to any type of algebraic-geometric code. This is mainly due to the algorithm's high decoding complexity and the mathematical explanation of the algorithm being not well defined. For example, to list decode of Hermitian codes, some parameters (corresponding coefficients of the code) of the codes are necessary for efficient implementation of the interpolation process. However, there is no suggested method on how to determine these parameters.

1.3 Aims and Objectives

This thesis aims to design an efficient decoding algorithm for both Reed-Solomon and algebraic-geometric codes and develop a software platform using the C programming language to evaluate the decoder's performance over both additive white Gaussian noise (AWGN) and Rayleigh fading channels. List decoding for algebraic-geometric codes is a new algorithm with better error-correction potential. In this thesis, both the hard-decision and soft-decision list decoding algorithms will be investigated for Reed-Solomon and algebraic-geometric codes. The achieved simulation results will be compared with the conventional unique decoding algorithms to show how much improvement can be gained.

The objectives of this research are:

- Reduce the decoding complexity for both hard-decision and soft-decision list decoding algorithms.
- Develop a soft-decision list decoding algorithm for algebraic-geometric codes.
- Develop a software platform for the efficient list decoder to evaluate its performance for both Reed-Solomon and algebraic-geometric codes.

1.4 Statement of Originality

This research project has investigated a complexity reduction scheme for the hard-decision list decoding of Reed-Solomon codes. This scheme reduces decoding complexity based on identifying and eliminating some unnecessary polynomials during the interpolation process. In fact, this is a general scheme which can also be applied to both soft-decision and hard-decision list decoding of Reed-Solomon and algebraic-geometric codes. Decoding complexity can be reduced by up to approximately 40%.

Further developing the mathematical framework for list decoding of Hermitian codes [9], this project produced the following modifications to it in order to obtain the first simulation results: First, an algorithm is proposed to determine the important parameters – corresponding coefficients of Hermitian codes. With the knowledge of these corresponding coefficients, the interpolation process can be efficiently implemented. Second, the developed complexity reduction scheme for interpolation is applied. Finally, based on the work of [10-12], a general factorisation algorithm is proposed, which can be efficiently implemented for both Reed-Solomon and algebraic-geometric codes.

This research project has also developed the first soft-decision list decoding algorithm for one of the best performing algebraic-geometric codes – Hermitian codes. It is shown that significant coding gains can be further achieved over the hard-decision list decoding scheme.

1.5 Organisation of the Thesis

The following chapters of the thesis are organised as follows: Chapter 2 will give a literature survey on the construction of algebraic-geometric codes, the conventional unique decoding algorithms and the list decoding algorithms which will be the main content of the thesis. Chapter 3 presents a theoretical background of this thesis, including some important parameters for constructing an algebraic-geometric code and the corresponding list decoding algorithm. Chapter 4 presents a hard-decision list decoding algorithm for Reed-Solomon codes introducing an original complexity reduction scheme while Chapter 5 presents a soft-decision list decoding algorithm for Reed-Solomon codes. Chapter 6 presents a hard-decision list decoding algorithm for Hermitian codes suggesting complexity reduction modifications while Chapter 7 presents the soft-decision list decoding algorithm for Hermitian codes. The conclusions of this thesis and some future research suggestions are presented in Chapter 8.

1.6 Publications Arising From This Project

So far, this project has resulted in 1 IET proceeding paper, 1 IET electronic letter and 3 conference papers being published. In addition, there is 1 IEEE transaction paper being accepted for publication and 1 IEEE transaction paper being submitted for review. These accepted and submitted publications are listed below as:

- L. Chen, R. A. Carrasco, and E. G. Chester, "Performance of Reed-Solomon codes using the Guruswami-Sudan algorithm with improved interpolation efficiency," *IET Commun*, vol. 1, pp. 241 - 250, 2007.
- L. Chen, R. A. Carrasco, and M. Johnston, "List decoding performance of algebraic geometric codes," *IET Electronic Letters*, vol. 42, 2006.
- L. Chen, R. A. Carrasco, M. Johnston, and E. G. Chester, "Efficient factorisation algorithm for list decoding algebraic-geometric and Reed-Solomon codes," presented at International Conference of Communications (ICC) 2007, Glasgow, UK, 2007.
- L. Chen, R. A. Carrasco, and E. G. Chester, "Decoding Reed-Solomon codes using the Guruswami-Sudan algorithm," presented at Communication Systems,

Networks, and Digital Signal Processing (CSNDSP) 2006, Patras Greece, 2006.

- L. Chen and R. A. Carrasco, "Efficient list decoder for algebraic-geometric codes," presented at 9th International Symposium on Communication Theory and Application (ISCTA'07), Ambleside, Lake district, UK, 2007.
- L. Chen, R. A. Carrasco, and M. Johnston, "Reduced complexity interpolation for list decoding Hermitian codes," *IEEE Trans. Wireless Commmun*, accepted for publication.
- L. Chen, R. A. Carrasco, and M. Johnston, "Soft-decision list decoding of Hermitian codes," *IEEE Trans. Commun*, Submitted for publication.

Chapter 2

Literature Survey

2.1 Introduction

This chapter presents a literature survey for the thesis. It starts from the construction of Reed-Solomon codes and general algebraic-geometric codes. Most of the important papers in the literature which give construction methods for general algebraic-geometric codes will be summarised in this chapter. Following that, the decoding methods for Reed-Solomon and algebraic-geometric codes will be briefly reviewed. There are mainly two types of decoding methods: unique decoding algorithms and list decoding algorithms. The unique decoding algorithms are conventional methods which are well developed and widely used nowadays. List decoding algorithms were only rediscovered in 1990s for their use with Reed-Solomon and algebraic-geometric codes. This type of algorithms tends to have greater performance than the unique decoding algorithms but with a higher complexity.

2.2 Construction of Reed-Solomon Codes and Algebraic-Geometric Codes

Reed-Solomon codes were introduced in the 1960s by Reed and Solomon [2]. They are non-binary block codes constructed from a generator polynomial defined over a finite field [13]. Reed-Solomon codes are widely used in wireless communication and storage systems and are still considered to be one of the most powerful error-correction codes. Goppa [1] introduced algebraic-geometric codes in the 1980s. Algebraic-geometric codes are constructed from an algebraic curve defined over a finite field. In fact, Reed-Solomon codes can be considered as a special case of algebraic-geometric codes constructed from a straight line.

The Gilbert-Varshamov bound [3, 13] defines a lower bound for a code's code rate $r = k/n$ and its relative minimum distance rate $\kappa = d/n$, where k, n and d are positive integers, and they are the dimension, length and minimum distance of the code respectively. Any code with parameters meeting this bound is said to be asymptotically good. Tsfasman, Vladut and Zink [14] presented method to construct asymptotically good algebraic-geometric codes from modular curves that exceed the Gilbert-Varshamov bound.

Justesen *et al* [15] presented a construction method for a class of algebraic-geometric codes which require simple algebraic geometry knowledge. Feng and Rao [16] later also presented a simple approach for construction of algebraic-geometric codes from affine plane curve. Following on, Xing *et al* [17] presented two constructions of linear codes from a local expansion of functions at a fixed rational point. They showed that their constructed codes have the same bound on their parameters as Goppa's geometry codes. Additionally, they showed linear codes constructed from the maximal curves can have better parameters than Goppa's geometry codes constructed from maximal curves. Heegard *et al* [18] showed how to construct systematic algebraic-geometric codes based on using the cyclic properties of automorphisms of the points on the curve. Blake *et al* [19] reviewed how to construct different algebraic-geometric codes constructed from different classes of curves, such as Klein quartic curve, elliptic curve, hyperelliptic curves and Hermitian curves.

2.3 Unique Decoding Algorithms

The conventional decoding algorithms for Reed-Solomon and algebraic-geometric codes result in one unique decoded message, the so-called unique decoding algorithms. The unique decoding algorithms are well developed and efficient in terms of running time. However, these unique decoding algorithms' error-correction capability is limited by the half-distance boundary $\left\lfloor \frac{d-1}{2} \right\rfloor$.

For Reed-Solomon codes, the most important unique decoding algorithms include: the Berlekamp-Massey decoding algorithm [3, 4], Euclid's decoding algorithm [20, 21], and the Peterson-Gorenstein-Zierler decoding algorithm [22, 23]. These algorithms first calculate the syndromes based on the received information. Then, error locations and error magnitudes are to be calculated in order to recover the corrected transmitted code word.

For algebraic-geometric codes, Justesen *et al* [15] gave the first decoding algorithm for codes derived from algebraic plane curve. This algorithm is a generalisation of

Peterson's algorithm [22]. However, the error-correction capability of the algorithm of [15] cannot reach the half designed distance bound. Skorobogatov and Vladut [24] generalised [15] for decoding codes arising from arbitrary algebraic curves. Especially for codes derived from elliptic and hyperelliptic curves, [24] can correct errors up to the half designed distance. Based on the Berlekamp-Massey algorithm [3, 4], Sakata [25] extended it to determine the error location using a two-dimensional array of syndromes which is suitable for applying to algebraic-geometric codes. Sakata later [26] extended [25] for higher dimensional arrays. Following on, Justesen *et al* [27] improved [25] for codes from an arbitrary regular plane curve. The improvement in [27] reduced the algorithm complexity to $O(n^{7/3})$. However, the above decoding algorithms still cannot reach the half designed distance bound for most algebraic-geometric codes.

Feng and Rao [6] presented a generalisation of the Peterson's algorithm [22] for algebraic-geometric codes. A majority voting scheme was introduced to determine the unknown syndromes so that the half designed distance bound is reached. The decoding complexity of Feng and Rao's algorithm is $O(n^3)$. Using Feng and Rao's majority voting scheme [6] and Sakata's generalisation [25] of the Berlekamp-Massey algorithm, Sakata [5] later presented a more efficient decoding algorithm, with complexity reduced to $O(n^{7/3})$. Based on the above work, given a received word with the number of errors not greater than the half designed distance bound, the error locations can be successfully determined. To determine the error magnitudes, Liu [28] presented a modified affine Fourier transform. Building upon the above literature, Johnston [29-31] investigated Hermitian code performance on the additive white Gaussian noise (AWGN) and fading channels, as well developing a clearer mathematical framework to make construction and decoding of algebraic-geometric codes more accessible. In [29] Hermitian codes can outperform similar code rate Reed-Solomon codes which are defined in the same finite field. Further, a basic soft-decision unique decoding algorithm for algebraic-geometric codes is presented in [32, 33] which introduced a fast erasure-and-error decoding algorithm. This algorithm's performance is later investigated by [34] showing further improvement can be achieved.

2.4 List Decoding Algorithms

The list decoding algorithm was first defined by Elias [35, 36] and Wozencraft [37] independently in the 1950s. The idea of the decoding algorithm is as follows: Given a received word R, reconstruct a list of code words with distance τ to R. Some later developments of the list decoding algorithm showed that τ can be greater than the half distance bound for both Reed-Solomon and algebraic-geometric codes, indicating better performance can be offered by the algorithm.

Building upon the work of Ar *et al* [38], Sudan [39, 40] proposed the first list decoding algorithm for Reed-Solomon codes which can correct errors beyond the half distance bound, provided the rate of the code is not greater than 1/3. Sudan's algorithm constructs an interpolated polynomial which passes through a set of points obtained from the received word and finds the transmitted message from the interpolated polynomial. In fact, the output transmitted message polynomial is the *root* of the interpolated polynomial. These two processes are called interpolation and factorisation respectively. Later, Roth and Ruckenstein [10] presented an efficient version of Sudan's algorithm, by which the same number of errors can be corrected and the complexity is reduced to $O(n^2\log^2 n)$. Another more important contribution of Roth and Ruckenstein's work [10] is an efficient method to implement the factorisation process of Sudan's algorithm, the recursive coefficient search method. Shokrollahi and Wasserman [41, 42] extended Sudan's algorithm to decode low rate algebraic-geometric codes. Also, they designed a factorisation algorithm that reduced the factorisation of polynomials defined over a larger finite field to the factorisation of polynomials defined over a smaller finite field.

The above list decoding algorithms for Reed-Solomon and algebraic-geometric codes can only correct errors beyond the half distance bound for low rate codes. In 1999, Guruswami and Sudan [7, 43] proposed an improved list decoding algorithm developed from the work of [39, 40]. The main contribution of Guruswami and Sudan's work is the definition of the interpolated polynomials as a polynomial that *intersects* a certain number of times over the set of points obtained from the received word R. As a result, the degree of the interpolated polynomials can be increased and so does the number of factorised outputs. By using Guruswami and Sudan's list decoding algorithm, almost all code rate Reed-Solomon codes and algebraic-geometric codes can be decoded beyond the half distance boundary. For implementing the interpolation process, an iterative polynomial construction algorithm [9, 44-47] can be applied. In fact, this process's complexity dominates the total list decoder complexity. Compared to interpolation, factorisation complexity is marginal. For implementation of factorisation, the most popular and efficient algorithm is the recursive coefficient search algorithm [10], which is later extended to factorise polynomials defined over the pole basis of an algebraic plane curve by [11, 12, 48]. Also, there are other alternatives, such as Shokrollahi and Wasserman [41]'s suggested method mentioned above, and Høholdt and Nielsen [9]'s suggested method that transfers the problem into factorising a univariate polynomial defined over a larger finite field.

To list decode one of the best performing algebraic-geometric codes – Hermitian codes, Høholdt and Nielsen [9, 44, 49] have presented a mathematical framework in terms of defining the *zero condition* of a polynomial defined over the pole basis of a Hermitian curve. However in the literature, there is still lack of knowledge on how to use Guruswami and Sudan's algorithm to decode other classes of algebraic-geometric codes.

One major development of Guruswami and Sudan's list decoding algorithm was achieved by Koetter and Vardy [8]. In [8], a soft-decision list decoding algorithm is presented for Reed-Solomon codes and its extension to algebraic-geometric codes seems obvious. Building upon the interpolation and factorisation processes, a priori process that converts the soft received information into interpolation information (*multiplicity*) is introduced. It is shown that this soft-decision scheme can offer significant coding gain compared to Guruswami and Sudan's hard-decision scheme.

Even though the list decoding algorithm can offer better performance than the conventional unique decoding algorithm, however, its complexity is higher and this probably is the reason why it is still not adopted in industrial applications. Therefore, any efficiency improved modification will be valuable in future research. One of the few suggestions was given by Koetter and Vardy [50] in which a transformation for the interpolation process was proposed.

2.5 Conclusion

This chapter has briefly reviewed the construction of Reed-Solomon and algebraic-geometric codes, as well as their decoding algorithms. For the decoding algorithms, it can be seen that the list decoding algorithm can offer better performance than the conventional and well developed unique decoding algorithm. List decoding algorithms could be the alternative choice for future industrial applications. One of the big challenges that lie ahead is on improving this decoding algorithm's efficiency. In the literature, there is still a lack of knowledge on how the list decoding algorithm performs for algebraic-geometric codes in different communication and storage environments. Therefore, further performance investigations seem to be valuable. Also, a soft-decision list decoding algorithm was only developed for Reed-Solomon codes. It is believed that greater performance improvement can be achieved by developing a soft-decision list decoding algorithm for algebraic-geometric codes.

Chapter 3

Theoretical Background

3.1 Introduction

This chapter presents the theoretical background of the thesis. It starts with a general description of algebraic-geometric codes. An algebraic-geometric code is constructed from an irreducible affine smooth curve. To define an algebraic-geometric code, we need to define the affine curve as well as the affine points and rational functions associated with the curve. Following that, two kinds of algebraic-geometric codes, Reed-Solomon codes and Hermitian codes, will be introduced as they are being investigated in the list decoding system which is described in the thesis. To clearly demonstrate the code construction process, two worked examples will be presented. For the purpose of supporting the list decoding system description, the pole basis and zero basis relating to these two codes will also be introduced. At the end of this chapter, a brief description of the list decoding algorithm and its application to Reed-Solomon codes and general algebraic-geometric codes will be presented.

3.2 Algebraic-Geometric Codes

An Algebraic-geometric code is constructed from an irreducible affine smooth curve [13, 19]. The construction of an algebraic-geometric code requires a set of points that satisfy the irreducible affine curve and a set of rational functions defined on the curve.

3.2.1 Projective and Affine Curves

A projective curve is a $(n + 1)$ dimensional curve, where n is a natural number, defined by projective points. Associated with this projective curve, there are $n + 1$ affine curves defined in different coordinate systems. For example, $\chi(x, y, z)$ is a 3 dimensional projective curve. Associated with it, there are $\chi(x, y, 1)$, $\chi(x, 1, z)$ and $\chi(1, y, z)$ affine curves.

If a curve cannot be expressed as a product of curves with lower degree, it is an irreducible curve. For example, $\chi(x, y) = x^3 + y^3$ is irreducible over GF(2). (Note: GF(q) denotes a Galois field with size q which is a prime number of a power of the prime number, its q elements can be written as: 0, 1, σ^1, σ^2, …, σ^{q-1}, where σ is a primitive element of the field.) A point is non-singular if not all partial derivatives of the curve vanish at this point. A curve is said to be smooth if all points on the curve are non-singular. One important class of irreducible smooth curves is the Hermitian curve. A Hermitian curve defined over GF(w^2) ($w = 2^{\mu}$, μ is a positive integer) can be written as:

$$\chi(x, y, z) = x^{w+1} + y^w z + yz^w \tag{3.1}$$

It is irreducible and it is smooth because partial derivatives $\frac{\partial\chi(x,y,z)}{\partial x} = (w + 1)x^w = x^w$, $\frac{\partial\chi(x,y,z)}{\partial y} = wy^{w-1}z + z^w = z^w$, and $\frac{\partial\chi(x,y,z)}{\partial z} = y^w + wyz^{w-1} = y^w$. The only point that makes all the three derivatives vanish is (0, 0, 0). However, (0, 0, 0) does not exist in projective space [13]. Therefore, all points are non-singular and the curve is smooth.

3.2.2 Points on an Affine Curve

The points (α, β, θ) that satisfies the projective curve $\chi(x, y, z) = 0$ are called projective points, where $\alpha, \beta, \theta \in$ GF(q). For construction of an algebraic-geometric code, an affine point of the form $p_i = (\alpha, \beta, 1)$ and a point at infinity of the form $p_\infty = (\alpha, \beta, 0)$ are needed. Codes constructed from curves with one point at infinity are called one-point algebraic-geometric codes or Goppa codes [13]. Reed-Solomon codes and Hermitian codes which are investigated in this thesis are classified as these codes. To find the affine points and the point at infinity, we need to define different affine components of the projective curve.

The Hasse-Weil bound [19] defines the number of points N (affine points and points at infinity) that satisfy a curve defined over GF(q) as:

$$|N| \leq (r - 1)(r - 2)\sqrt{q} + 1 + q \tag{3.2}$$

where r is the degree of the curve. Curves that meet this bound are called maximal curves.

Here two case studies are shown to find affine points and the point at infinity.

Case study 1: Find the points on the straight line $y = 0$ defined over GF(4).

σ is a primitive element in GF(4) that satisfies $\sigma^2 + \sigma + 1 = 0$. Addition and multiplication table of GF(4) is shown in Appendix A.

In the $(x, y, 1)$ system, projective points are:

$p_0 = (0, 0, 1)$	$p_1 = (1, 0, 1)$	$p_2 = (\sigma, 0, 1)$	$p_3 = (\sigma^2, 0, 1)$

Table 3.1 projective points on $y = 0$ in the $(x, y, 1)$ system

In the $(x, 1, y)$ system, there is no projective points as $y = 0$.

In the $(1, y, z)$ system, the projective points are:

$p_0 = (1, 0, 0)$	$p_1 = (1, 0, 1)$	$p_2 = (1, 0, \sigma)$	$p_3 = (1, 0, \sigma^2)$

Table 3.2 projective points on $y = 0$ in the $(1, y, z)$ system

Therefore, on line $y = 0$, there are 4 affine points as: $p_0 = (0, 0, 1)$, $p_1 = (1, 0, 1)$, $p_2 = (\sigma, 0, 1)$, $p_3 = (\sigma^2, 0, 1)$, and 1 point at infinity as: $p_\infty = (1, 0, 0)$. For the straight line $y = 0$ defined over GF(4), Hasse-Weil bound is $|N| \leq 1+ 4 = 5$. Therefore, line $y = 0$ is a maximal curve.

Case study 2: Find the points on Hermitian curve $x^3 + y^2z + yz^2 = 0$ define over GF(4).

In the $(x, y, 1)$ system, projective points are:

$p_0 = (0, 0, 1)$	$p_1 = (0, 1, 1)$	$p_2 = (1, \sigma, 1)$	$p_3 = (1, \sigma^2, 1)$
$p_4 = (\sigma, \sigma, 1)$	$p_5 = (\sigma, \sigma^2, 1)$	$p_6 = (\sigma^2, \sigma, 1)$	$p_7 = (\sigma^2, \sigma^2, 1)$

Table 3.3 projective points on $x^3 + y^2z + yz^2 = 0$ in the $(x, y, 1)$ system

In the $(x, 1, z)$ system, projective points are:

$p_0 = (0, 1, 0)$	$p_1 = (0, 1, 1)$	$p_2 = (1, 1, \sigma)$	$p_3 = (1, 1, \sigma^2)$
$p_4 = (\sigma, 1, \sigma)$	$p_5 = (\sigma, 1, \sigma^2)$	$p_6 = (\sigma^2, 1, \sigma)$	$p_7 = (\sigma^2, 1, \sigma^2)$

Table 3.4 projective points on $x^3 + y^2z + yz^2 = 0$ in the $(x, 1, z)$ system

In the (1, y, z) system, projective points are:

$p_0 = (1, \sigma, 1)$	$p_1 = (1, \sigma, \sigma^2)$	$p_2 = (1, \sigma^2, 1)$	$p_3 = (1, \sigma^2, \sigma)$

Table 3.5 projective points on $x^3 + y^2z + yz^2 = 0$ in the (1, y, z) system

Therefore, on curve $x^3 + y^2z + yz^2 = 0$, there are 8 affine points as: $p_0 = (0, 0, 1)$, $p_1 = (0, 1, 1)$, $p_2 = (1, \sigma, 1)$, $p_3 = (1, \sigma^2, 1)$, $p_4 = (\sigma, \sigma, 1)$, $p_5 = (\sigma, \sigma^2, 1)$, $p_6 = (\sigma^2, \sigma, 1)$, $p_7 = (\sigma^2, \sigma^2, 1)$, and 1 point at infinity $p_\infty = (0, 1, 0)$. For this Hermitian curve, the Hasse-Weil bound is: $|N| \le (3 - 1)(3 - 2)\sqrt{4} + 1 + 4 = 9$ and it is a maximal curve.

The above two case studies illustrate that over the same Galois field, there are more points on a Hermitian curve than on a straight line. This enables Hermitian codes to have longer code lengths than the Reed-Solomon codes which are constructed from a straight line.

3.2.3 Rational Functions on the Curves

Rational functions are a quotient of two other functions both of which have the same degree as: $f(x, y, z) = \dfrac{g(x,y,z)}{h(x,y,z)}$. The order of rational function $f(x, y, z)$ (denoted as $v(f(x, y, z))$) at a point is a sum of its zero orders and pole orders [13]. To construct an algebraic-geometric code, each rational function is evaluated at the set of affine points to form a row of the generator matrix, which will be described in later of this chapter. These rational functions should have a pole at the point of infinity but not other affine points. Again, here gives two case studies to define that rational functions on the straight line $y = 0$ and Hermitian curve $x^{w+1} + y^w z + yz^w = 0$.

Case study 3: Define the rational functions on straight line $y = 0$.

The set of rational functions defined on $y = 0$ can be written as: $\left\{\dfrac{x^i}{z^i}\right\}$, $i \ge 0$. Based on case study 1, the point at infinity on $y = 0$ is $p_\infty = (1, 0, 0)$, $\dfrac{x^i}{z^i}$ has pole at p_∞ as $z = 0$.

However, as all the affine points has z = 1, $\frac{x^i}{z^i}$ has not pole at all the affine points. It is easy to realise that $\frac{x^i}{z^i}$ has no zero orders at p_∞ since x = 1, but pole order i. Therefore, rational functions $\frac{x^i}{z^i}$ have increasing orders at p_∞ as: $v(\frac{x^i}{z^i}) = i$.

Case study 4: Define rational functions on Hermitian curve $x^{w+1} + y^w z + yz^w = 0$.

The set of rational functions defined on the Hermitian curve can be generally written as: $\frac{x^i y^j}{z^{i+j}}$, $0 \le i \le w$ and $j \ge 0$ [19, 51]. Based on case study 2, the point at infinity on this Hermitian curve is p_∞ = (0, 1, 0). Rational function $\frac{x^i y^j}{z^{i+j}}$ has pole at p_∞ as z = 0 but not other affine points as for other affine points z = 1. To define functions $\frac{x^i y^j}{z^{i+j}}$'s order at p_∞, it is important to realise that:

$$\frac{x}{z} = \frac{y^w + yz^{w-1}}{x^w} \text{ and } \frac{y}{z} = \frac{y^{w+1} + y^2 z^{w-1}}{x^{w+1}}$$

As for p_∞, x = 0 and y = 1, $\frac{x}{z}$ has no zero order at p_∞, but has pole order w, while $\frac{y}{z}$ has no zero order at p_∞, but has pole order w + 1. Therefore, at p_∞, $v(\frac{x}{z}) = w$ and $v(\frac{y}{z}) = w + 1$. Therefore, given a general rational function $\frac{x^i y^j}{z^{i+j}}$, its order at p_∞ is: $v(\frac{x^i y^j}{z^{i+j}}) = iw + j(w + 1)$.

A devisor of a curve assigns an integer value to every point on the curve. An algebraic-geometric code is defined by two kinds of devisor: divisors D and G. Devisor D assigns a value $D(p_i)$ = 1 to every affine point and is a sum of all the affine points [19]:

$$D = \sum_{i=0}^{n-1} D(p_i)\, p_i = \sum_{i=0}^{n-1} p_i \tag{3.3}$$

$\sum_{i=0}^{n-1} D(p_i)$ is the degree of D, denoted as $d(D)$. Devisor G assigns an integer value $D(p_\infty)$ to the point at infinity p_∞. For curves with one point at infinity:

$$G = d(G)\, p_\infty \tag{3.4}$$

where $d(G) = D(p_\infty)$. Therefore, $L(G)$ defines the sequence of rational functions with order at p_∞ not greater than $d(G)$. Take the rational functions shown by case study 4 as an example, if $d(G) = 13$, then

$$L(G) = L(13\, p_\infty) = \left\{ \frac{1}{z^0}, \frac{x}{z}, \frac{y}{z}, \frac{x^2}{z^2}, \frac{xy}{z^2}, \frac{y^2}{z^2}, \frac{x^2y}{z^3}, \frac{xy^2}{z^3}, \frac{y^3}{z^3}, \frac{x^2y^2}{z^4}, \frac{xy^3}{z^4}, \frac{y^4}{z^4}, \frac{x^2y^3}{z^5} \right\}.$$

3.2.4 Construction of Algebraic-Geometric Codes

Based on the above study, it is sufficient to define an algebraic-geometric code's parameters and its construction method.

The Riemann-Roch theorem [13] defines the number of rational functions in $L(G)$ with order at p_∞ not greater than $d(G)$, and therefore defines the dimension of the code, l(G). Based on the Riemann-Roch theorem, there exists a nonnegative integer g that:

$$l(G) - d(G) = 1 - g \tag{3.5}$$

given $d(G) > 2g - 2$. The nonnegative integer g is called the genus of the curve, defined as:

$$g = \frac{(r-1)(r-2)}{2} \tag{3.6}$$

where r is the degree of the curve. The nonnegative integers that match the order numbers of rational functions in $L(G)$ are called nongaps. Otherwise, they are gaps. The maximal number of gaps is g.

For constructing an $(n,\ k)$ algebraic-geometric code, the message length k is the dimension of $L(G)$. Based on (3.5),

$$k = l(G) = d(G) + 1 - g \tag{3.7}$$

The code word length n is decided by the number of affine points n. This code has designed minimal distance d^*:

$$d^* = n - k - g + 1 \tag{3.8}$$

When $g = 0$, d^* becomes the optimal Hamming distance:

$$d = n - k + 1 \tag{3.9}$$

Reed-Solomon codes have optimal Hamming distance as it is constructed from a straight line with genus $g = 0$. Compared with it, Hermitian codes suffer from genus penalty. However, Hermitian codes have larger designed minimal distance as there are more affine points on a Hermitian curve than on a straight line.

A $k \times n$ generator matrix is formed to construct a (n, k) algebraic-geometric codes. In the generator matrix, k rows are formed by evaluating the k rational function in $L(G)$ ($d(G) = k - 1 + g$) over the n affine points. Then, a code word vector with length n is generated by multiplying a message vector with length k to the generator matrix. The construction of Reed-Solomon codes and Hermitian codes will be described in the following sections 3.3 and 3.4 respectively.

3.3 Reed-Solomon Codes

As a special kind of algebraic-geometric codes, Reed-Solomon codes are constructed from a straight line $y = 0$ [2]. Based on case study 1, the affine points $p_i(x, y, z)$ on a straight line has $y = 0$ and $z = 1$, and they can be distinctively denoted by their x-coordinates (finite field elements). The rational functions $\frac{x^i}{z^i}$ ($i \geq 0$) introduced in case study 3 can be simplified as: x^i ($i \geq 0$). Therefore, the sequence of rational functions for Reed-Solomon codes can be defined as:

$$L(\infty p_\infty) = \{1, x, x^2, x^3, x^4, \ldots\ldots\} \tag{3.10}$$

$L(Sp_\infty)$ is a subset of $L(\infty p_\infty)$ with order of functions in $L(\infty p_\infty)$ not greater than the nonnegative integer S. For construction of a (n, k) ($k < n$) Reed-Solomon code defined over GF(q) ($n = q$ - 1), the generator matrix G_{RS} can be formed by evaluating the first k functions of (3.10) at n finite field elements $x_0, x_1, \dots, x_{n-1} \in \mathrm{GF}(q)\backslash\{0\}$ as:

$$G_{RS} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_{n-1} \\ \vdots & & \ddots & \\ x_0^{k-1} & x_1^{k-1} & \cdots & x_{n-1}^{k-1} \end{bmatrix} \tag{3.11}$$

Then the message vector $\bar{f} = (f_0, f_1, \dots, f_{k-1}) \in \mathrm{GF}(q)$ is multiplied to G_{RS} to generate a code word as:

$$\bar{c} = (c_0, c_1, \dots, c_{n-1}) = \bar{f} \times G_{RS} \tag{3.12}$$

The encoding process can also be equivalently described in a polynomial evaluation manner. Defining the message polynomial $f(x)$ as:

$$f(x) = f_0 \cdot 1 + f_1 \cdot x + \cdots + f_{k-1} \cdot x^{k-1} \tag{3.13}$$

where coefficients $f_0, f_1, \dots, f_{k-1} \in \mathrm{GF}(q)$ are message symbols and $1, x, \dots, x^{k-1}$ are the rational functions in $L((k-1)p_\infty)$. Code word $\bar{c}$ is generated by evaluating $f(x)$ over the n finite field elements as:

$$\bar{c} = (c_0, c_1, \dots, c_{n-1}) = (f(x_0), f(x_1), \dots, f(x_{n-1})) \tag{3.14}$$

In the list decoding system, this encoding manner is used and so as in the following description of the thesis.

Based on (3.9), a (n, k) Reed-Solomon code has Hamming distance $d = n - k + 1$. This code has error-correction capability:

$$\tau = \left\lfloor \frac{d-1}{2} \right\rfloor \tag{3.15}$$

This is the error-correction bound applied to the conventional unique decoding algorithms [3, 4, 20-22] for Reed-Solomon codes. However, in later of this thesis, the list decoding algorithm can perform beyond this bound.

3.3.1 Example: Construct a (15, 9) Reed-Solomon Code Defined in GF(16)

σ is a primitive element in GF(16) that satisfies $\sigma^4 + \sigma + 1 = 0$. Addition and multiplication table of GF(16) is shown in Appendix C.

Given the message polynomial $f(x)$ as:

$f(x) = \sigma + \sigma^4x + \sigma^4x^2 + \sigma^2x^3 + \sigma^7x^4 + \sigma^{13}x^5 + \sigma^6x^6 + 1\,x^7 + \sigma^6x^8$

The 15 finite field elements in GF(16)\{0} are:

$(x_0, x_1, \ldots, x_{14}) = (1, \sigma, \sigma^4, \sigma^2, \sigma^8, \sigma^5, \sigma^{10}, \sigma^3, \sigma^{14}, \sigma^9, \sigma^7, \sigma^6, \sigma^{13}, \sigma^{11}, \sigma^{12})$.

Evaluate them in $f(x)$, we have:

$c_0 = f(x_0) = f(1) = 1$, $c_1 = f(x_1) = f(\sigma) = \sigma^8$, $c_2 = f(x_2) = f(\sigma^4) = \sigma^8$, $c_3 = f(x_3) = f(\sigma^2) = 0$,

$c_4 = f(x_4) = f(\sigma^8) = \sigma^{14}$, $c_5 = f(x_5) = f(\sigma^5) = \sigma^{10}$, $c_6 = f(\mathrm{x}_6) = f(\sigma^{10}) = \sigma^6$,

$c_7 = f(x_7) = f(\sigma^3) = \sigma^{14}$, $c_8 = f(x_8) = f(\sigma^{14}) = \sigma^4$, $c_9 = f(x_9) = f(\sigma^9) = \sigma^6$,

$c_{10} = f(x_{10}) = f(\sigma^7) = 0$, $c_{11} = f(x_{11}) = f(\sigma^6) = \sigma^2$, $c_{12} = f(x_{12}) = f(\sigma^{13}) = \sigma^5$,

$c_{13} = f(x_{13}) = f(\sigma^{11}) = \sigma^9$, $c_{14} = f(x_{14}) = f(\sigma^{12}) = \sigma^{12}$.

Therefore, the code word $\bar{c} = (1, \sigma^8, \sigma^8, 0, \sigma^{14}, \sigma^{10}, \sigma^6, \sigma^{14}, \sigma^4, \sigma^6, 0, \sigma^2, \sigma^5, \sigma^9, \sigma^{12})$.

The error-correction bound for this code is $\tau = 3$.

3.3.2 Pole Basis and Zero Basis

Pole basis contains a set of monomials with increasing pole orders at the point of infinity p_∞. Based on the above description, the pole basis is defined by the sequence of rational functions $L(\infty p_\infty)$ (3.10). Pole basis is introduced because it defines the polynomials associated with the corresponding algebraic curves when performing Guruswami-Sudan's list decoding algorithm [7, 43]. For Reed-Solomon codes, polynomials introduced in the algorithm can generally be written as: $f(x, y) = \sum_{a,b\in \mathrm{N}} f_{ab}x^a y^b$, where $f_{ab} \in \mathrm{GF}(q)$.

With respect to every finite field element, there also exists a basis of functions ψ with increasing zero orders at p_i, $v_{p_i}(\psi)$. $v_{p_i}(\psi)$ can be evaluated by dividing ψ by $(x - x_i)$ until a unit (a function which evaluates to a non-zero value at x_i) is obtained. The zero order is equal to the number of division in order to obtain the unit. In general, with respect to x_i, ψ can be written as:

$$\psi = (x - x_i)^{\alpha} \quad (\alpha \in \mathrm{N}) \tag{3.16}$$

It is easy to realise that, $v_{p_i}(\psi) = \alpha$ as unit (evaluation value equals to 1) can be obtained after α divisions.

Pole basis and zero basis functions are introduced because they are used to define a polynomial's zero conditions (the singularity of the interpolated unit) in the list decoding algorithm.

3.4 Hermitian Codes

Hermitian codes are constructed from Hermitian curves $x^{w+1} + y^w + y = 0$. Based on case study 2, the affine points have z-coordinate equals to 1 and they can be distinctively denotes as: $p_0 = (x_0, y_0)$, $p_1 = (x_1, y_1)$, …, and $p_{n-1} = (x_{n-1}, y_{n-1})$, where $n = w^3$. From case study 4, we also know that the rational functions for Hermitian codes can be simply denoted as: $x^i y^j$ ($0 \le i \le w$ and $j \ge 0$). Therefore, the sequence $L(\infty p_\infty)$ for Hermitian codes can be written as:

$$L(\infty p_\infty) = \{\phi(x, y) \mid \phi(x, y) = 1, x, y, \ldots, x^i y^j, \ldots, 0 \le i \le w \text{ and } j \ge 0\} \tag{3.17}$$

In order to distinguish different rational functions sequence associated with different Hermitian curve, we denote the rational functions sequence associated with curve $x^{w+1} + y^w + y = 0$ as $L_w(\infty p_\infty)$. Here gives two examples showing difference rational functions sequences derived from different Hermitian curves.

Example 3.1 For Hermitian curve $x^3 + y^2 + y = 0$, $w = 2$ and $L_2(\infty p_\infty) = \{1, x, y, x^2, xy, y^2, x^2y, xy^2, y^3, x^2y^2, xy^3, y^4, \ldots\}$.

Example 3.2 For Hermitian curve $x^5 + y^4 + y = 0$, $w = 4$ and $L_4(\infty p_\infty) = \{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, y^4, x^4y, x^3y^2, x^2y^3, xy^4, y^5, \ldots\}$.

$L_w(Sp_\infty)$ is a subset of $L_w(\infty p_\infty)$ with order of functions in $L(\infty p_\infty)$ not greater than the nonnegative integer S. For example, $L_2(9p_\infty) = \{1, x, y, x^2, xy, y^2, x^2y, xy^2, y^3\}$.

For the construction of a (n, k) Hermitian code from curve $x^{w+1} + y^w + y = 0$, the k rational functions in $L_w(l\,p_\infty)$ (given $l > 2g - 1$, $k = l - g + 1$) are evaluated over the n affine points to form a $k \times n$ generator matrix as:

$$G_{Herm} = \begin{bmatrix} \phi_0(p_0) & \phi_0(p_1) & \cdots & \phi_0(p_{n-1}) \\ \phi_1(p_0) & \phi_1(p_1) & \cdots & \phi_1(p_{n-1}) \\ \vdots & & \ddots & \\ \phi_{k-1}(p_0) & \phi_{k-1}(p_1) & \cdots & \phi_{k-1}(p_{n-1}) \end{bmatrix} \tag{3.18}$$

Then the message vector $\bar{f} = (f_0, f_1, \ldots, f_{k-1}) \in \mathrm{GF}(q)$ is multiplied to G_{Herm} to generate a code word as:

$$\bar{c} = (c_0, c_1, \ldots, c_{n-1}) = \bar{f} \times G_{Herm} \tag{3.19}$$

To describe the encoding process in a polynomial evaluation manner, we have the message polynomial f written as:

$$f(x, y) = f_0 \cdot \phi_0 + f_1 \cdot \phi_1 + \cdots + f_{k-1} \cdot \phi_{k-1} \tag{3.20}$$

and the code word is generated as:

$$\bar{c} = (c_0, c_1, \ldots, c_{n-1}) = (f(p_0), f(p_1), \ldots, f(p_{n-1})) \tag{3.21}$$

Based on (3.8), this Hermitian code has designed minimal distance $d^* = n - k - g + 1$ and its error-correction capability for the conventional unique decoding algorithm [5, 6, 25] is defined as:

$$\tau = \left\lfloor \frac{d^* - 1}{2} \right\rfloor \tag{3.22}$$

Do notice that for high rate Hermitian codes, their minimal distance is greater than the designed distance and therefore (3.22) is not a tight bound for those codes.

3.4.1 Example: Construct a (8, 4) Hermitian Code Defined in GF(4)

The Hermitian curve defined in GF(4) and its associated rational functions are given in example 3.1. On this curve, the 8 affine points are: $p_0 = (0, 0)$, $p_1 = (0, 1)$, $p_2 = (1, \sigma)$, $p_3 = (1, \sigma^2)$, $p_4 = (\sigma, \sigma)$, $p_5 = (\sigma, \sigma^2)$, $p_6 = (\sigma^2, \sigma)$ and $p_7 = (\sigma^2, \sigma^2)$.

The message polynomial is given as: $f(x, y) = 1 + \sigma x + y + \sigma^2 x^2$.

Evaluating $f(x, y)$ over the 8 affine points, we have:

$c_0 = f(p_0) = f(0, 0) = 1$, $c_1 = f(p_1) = f(0, 1) = 0$, $c_2 = f(p_2) = f(1, \sigma) = \sigma$,

$c_3 = f(p_3) = f(1, \sigma^2) = \sigma^2$, $c_4 = f(p_4) = f(\sigma, \sigma) = \sigma$, $c_5 = f(p_5) = f(\sigma, \sigma^2) = \sigma^2$,

$c_6 = f(p_6) = f(\sigma^2, \sigma) = \sigma^2$, and $c_7 = f(p_7) = f(\sigma^2, \sigma^2) = \sigma$.

Therefore, the code word $\bar{c} = (1, 0, \sigma, \sigma^2, \sigma, \sigma^2, \sigma^2, \sigma)$.

The designed minimal distance for the code is $d^* = 4$ and its error-correction capability is $\tau = 1$.

3.4.2 Pole Basis and Zero Basis

The sequence of rational functions $L_w(\infty p_\infty)$ defines the pole basis monomials associated with Hermitian curve $x^{w+1} + y^w + y = 0$. These pole basis monomials have increasing pole order at the point of infinity p_∞ as:

$$L_w(\infty p_\infty) = \{\phi_a(x, y) \mid v_{p_\infty}(\phi_a^{-1}) < v_{p_\infty}(\phi_{a+1}^{-1}),\ a \in \mathrm{N}\} \tag{3.23}$$

When applying the Guruswami-Sudan list decoding algorithm [7, 43], polynomials can be generally written as: $f(x, y, z) = \sum_{a,b \in \mathrm{N}} f_{ab}\phi_a(x, y)z^b$, where $f_{ab} \in \mathrm{GF}(q)$.

With respect to every affine point $p_i = (x_i, y_i)$, there also exists a zero basis which contains rational functions with increasing zero orders at p_i as [9]:

$$Z_{w,p_i} = \{\psi_{p_i,\alpha}(x,y) \mid v_{p_i}(\psi_{p_i,\alpha}) < v_{p_i}(\psi_{p_i,\alpha+1}),\ \alpha \in \mathrm{N}\} \qquad (3.24)$$

function $\psi_{p_i,\alpha}$ has zero order $v_{p_i}(\psi_{p_i,\alpha}) = \alpha$ at p_i. To evaluate the zero order, $\psi_{p_i,\alpha}$ is divided by $(x - x_i)$ until a unit has been obtained. Again, the zero order is equal to the number of divisions. In general, $\psi_{p_i,\alpha}$ can be written as [44]:

$$\psi_{p_i,\alpha}(x,y) = \psi_{p_i,\lambda+(w+1)\delta}(x,y) = (x-x_i)^{\lambda}[(y-y_i) - x_i^{w}(x-x_i)]^{\delta} \qquad (3.25)$$

where λ, $\delta \in \mathrm{N}$, $0 \le \lambda \le w$ and $\delta \ge 0$. In the following, example 3.3 lists some zero basis functions with respect to an affine point. Example 3.4 illustrates how to evaluate the zero order of functions in (3.25).

Example 3.3 Given $p_i = (1, \sigma)$ as an affine point on curve $x^3 + y^2 + y = 0$, list the first 8 zero basis functions with respect to this point.

Based on (3.25), $\psi_{p_i,\alpha}(x,y) = \psi_{p_i,\lambda+3\delta}(x,y) = (x-1)^{\lambda}[(y-\sigma) - 1^2(x-1)]^{\delta}$, where λ, $\delta \in \mathrm{N}$, $0 \le \lambda \le 2$ and $\delta \ge 0$. Therefore,

$\psi_{p_i,0}(x,y) = (x-1)^0[(y-\sigma) - 1^2(x-1)]^0 = 1$

$\psi_{p_i,1}(x,y) = (x-1)^1[(y-\sigma) - 1^2(x-1)]^0 = 1 + x$

$\psi_{p_i,2}(x,y) = (x-1)^2[(y-\sigma) - 1^2(x-1)]^0 = 1 + x^2$

$\psi_{p_i,3}(x,y) = (x-1)^0[(y-\sigma) - 1^2(x-1)]^1 = \sigma^2 + x + y$

$\psi_{p_i,4}(x,y) = (x-1)^1[(y-\sigma) - 1^2(x-1)]^1 = \sigma^2 + \sigma x + y + x^2 + xy$

$\psi_{p_i,5}(x,y) = (x-1)^2[(y-\sigma) - 1^2(x-1)]^1 = \sigma^2 + x + \sigma^2x^2 + y^2 + x^2y$

$\psi_{p_i,6}(x,y) = (x-1)^0[(y-\sigma) - 1^2(x-1)]^2 = \sigma + x^2 + y^2$

$\psi_{p_i,7}(x,y) = (x-1)^1[(y-\sigma) - 1^2(x-1)]^2 = \sigma + \sigma x + y + x^2 + xy^2$.

Example 3.4 Based on the example 3.3, justify zero basis function $\psi_{p_i,3}$ has zero order 3 at p_i.

To evaluate a function's zero order at an affine point of Hermitian curve $x^{w+1} + y^w + y = 0$, it is important to notice the following equation associated with the curve [9]:

$$\frac{y - y_i}{x - x_i} = \frac{(x - x_i)^w + x_i(x - x_i)^{w-1} + x_i^{\ w}}{e} \tag{3.26}$$

where $e = (y - y_i)^{w-1} + 1$. It can be seen that $e(p_i) = (y_i - y_i)^{w-1} + 1 = 1$.

From example 3.3, it can be seen that $\psi_{p_i,3}(x, y) = (y - \sigma) - (x - 1)$ and $e = (y - \sigma) + 1$.

Initialise $\psi^{(0)}(x, y) = \psi_{p_i,3}(x, y) = (y - \sigma) - (x - 1)$.

The 1st division: $\psi^{(1)}(x, y) = \dfrac{\psi^{(0)}(x,y)}{x-1} = \dfrac{y-\sigma}{x-1} - 1 = \dfrac{(x-1)^2 + (x-1) + 1}{e} - 1$

$= \dfrac{(x-1)^2 + (x-1) + 1 - (y-\sigma) - 1}{e} = (x - 1)e^{-1} + (y - \sigma)e^{-1} + (x - 1)^2e^{-1}$.

We have $\psi^{(1)}(p_i) = (1 - 1)\cdot 1 + (\sigma - \sigma)\cdot 1 + (1 - 1)^2\cdot 1 = 0$.

The 2nd division: $\psi^{(2)}(x, y) = \dfrac{\psi^{(1)}(x,y)}{x-1} = e^{-1} - \dfrac{y-\sigma}{x-1}e^{-1} + (x - 1)e^{-1} = e^{-1} - [(x - 1)^2 + (x - 1) + 1]e^{-2} + (x - 1)e^{-1} = (e^{-1} - e^{-2}) - (x - 1)(e^{-2} - e^{-1}) - (x - 1)^2e^{-2}$.

We have $\psi^{(2)}(p_i) = (1 - 1) - (1 - 1)\cdot(1 - 1) - (1 - 1)^2\cdot 1 = 0$.

The 3rd division: $\psi^{(3)}(x, y) = \dfrac{\psi^{(2)}(x,y)}{x-1} = \dfrac{e^{-1} - e^{-2}}{x-1} - (e^{-2} - e^{-1}) - (x - 1)e^{-2} = \dfrac{e-1}{(x-1)e^2} - (e^{-2} - e^{-1}) - (x - 1)e^{-2} = \dfrac{y-\sigma}{x-1}e^{-2} - (e^{-2} - e^{-1}) - (x - 1)e^{-2} = [(x - 1)^2 + (x - 1) + 1]e^{-3} - (e^{-2} - e^{-1}) - (x - 1)e^{-2} = (e^{-3} - e^{-2} + e^{-1}) + (x - 1)(e^{-3} - e^{-2}) + (x - 1)^2e^{-3}$.

We have $\psi^{(3)}(p_i) = (1 - 1 + 1) + (1 - 1)\cdot(1 - 1) + (1 - 1)^2\cdot 1 = 1 \neq 0$.

There are 3 divisions in order obtain a unit. Therefore, the zero order of $\psi_{p_i,3}$ at p_i is 3 as: $v_{p_i}(\psi_{p_i,3}) = 3$. $\psi_{p_i,3}$ can also be written as: $\psi_{p_i,3} = (x - 1)^3[(e^{-3} - e^{-2} + e^{-1}) + (x - 1)(e^{-3} - e^{-2}) + (x - 1)^2e^{-3}]$. A general algorithm for evaluating the zero basis functions is presented in [9].

3.5 List Decoding

The conventional unique decoding algorithms for Algebraic-geometric and Reed-Solomon codes are efficient in terms of running time, but with error-correction capability limited by the half distance boundary τ which is defined by (3.15) for Reed-Solomon codes and (3.22) for algebraic-geometric codes. As introduced in Chapter 2, the list decoding algorithm is a newly rediscovered method which can correct errors beyond the half distance boundary. This section gives a brief introduction to the list decoding algorithm. In this section, the unique decoding algorithm's bound τ is denoted as τ_{unique} in order to avoid conflicting notation when we introduce the error-correction bound for list decoding.

3.5.1 The Idea of List Decoding

Elias [35, 36] and Wozencraft [37] first introduced the idea of list decoding which leads to the later solution of decoding Reed-Solomon and algebraic-geometric codes beyond boundary τ_{unique}. Their idea can be described as: given a received word R, reconstruct a list of all code words with a distance τ to the received word R, in which τ can be greater than τ_{unique}. This idea can be illustrated by Fig 3.1. In Fig 3.1, $c1$, $c2$, and $c3$ are 3 independent code words with distance d to each other (for algebraic-geometric codes, d is the designed minimal distance (3.8). For Reed-Solomon codes, d is the Hamming distance (3.9)). For received word $r1$ which has distance less than $\left\lfloor \frac{d-1}{2} \right\rfloor$ to code word $c1$, it can be decoded by the unique decoding algorithm which results in $c1$. However, for received word $r2$ which has distance greater than $\left\lfloor \frac{d-1}{2} \right\rfloor$ to any of the code word, the unique decoding algorithm will fail to decode it. However, using the list decoding algorithm, a list of possible transmitted code word will be produced. For example, decoded output list $\{c1, c2, c3\}$ is produced by the decoder. Then, the code word that has the minimal distance to $r2$ is chosen from the list and decoding is completed.

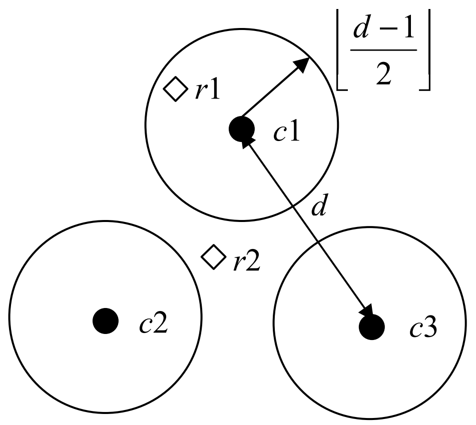


Figure 3.1 Idea of list decoding

3.5.2 List Decoding of Low Rate Reed-Solomon and Algebraic-Geometric Codes

Sudan [39, 40] introduced the first list decoding algorithm for low rate ($k/n < 1/3$) Reed-Solomon codes. For a (n, k) Reed-Solomon code, given the received word $R = (r_0, r_1, \ldots, r_{n-1})$ ($r_i \in \text{GF}(q)$, $i = 0, 1, \ldots, n - 1$), n interpolated units can be formed by combining received symbol r_i with the respective finite field element x_i used in encoding (3.14) as: $\{(x_0, r_0), (x_1, r_1), \ldots, (x_{n-1}, r_{n-1})\}$. The first step of the algorithm is to find polynomial $Q(x, y)$ that passes through these n interpolated units as: $Q(x_i, y_i) = 0$. The second step of the algorithm is to find polynomials $f(x)$ with degree less than k and $f(x_i) = r_i$ for at least $n - \tau$ values. $y - f(x)$ is a factor of $Q(x, y)$ as: $y - f(x) \mid Q(x, y)$ or $Q(x, f(x)) = 0$. If $f(x)$ is the transmitted message polynomial (3.13), then τ errors in received word R can be corrected. This process can be geometrically illustrated by Fig 3.2. For a (5, 2) Reed-Solomon code, 5 interpolated units are geometrically presented in the figure. The polynomial to be found in the first step is $Q(x, y) = y^2 - x^2$. As $Q(x, y) = y^2 - x^2 = (y + x)(y - x)$. Therefore, in the second step, the output polynomial $f(x) = -x$ or $f(x) = x$. From Fig 3.2, it can be seen that $f(x) = x$ satisfies $f(x_i) = r_i$ for (x_1, r_1), (x_2, r_2), and (x_3, r_3), while $f(x) = -x$ satisfies $\text{f}(x_\text{i}) = r_\text{i}$ for (x_0, r_0), (x_2, r_2) and (x_4, r_4). The algorithm corrects $\tau = 2$ errors.

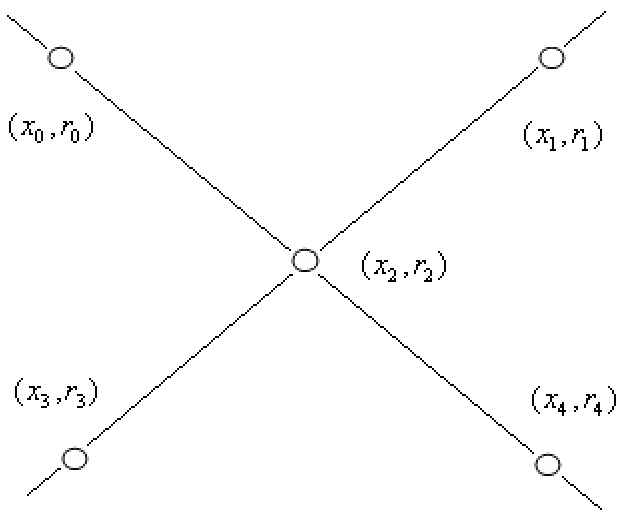


Figure 3.2 Geometric illustration of list decoding

Shokrollahi and Wasserman [41, 42] extended Sudan's work [39, 40] for list decoding of low rate algebraic-geometric codes. For a (n, k) algebraic-geometric code, given received word R, n interpolated units can be formed by combining each received symbol r_i with the respective affine point p_i used in encoding: $\{(p_0, r_0), (p_1, r_1), \ldots, (p_{n-1}, r_{n-1})\}$. The first step is the find polynomial $Q(x, y, z)$ that passes through (p_i, r_i) (where $p_i = (x_i, y_i)$) as: $Q(x_i, y_i, r_i) = 0$. The second step is find polynomial $f(x, y)$ defined in $L(l\ p_\infty)$ ($l = k + g - 1$ and p_∞ is a point at infinity on the corresponding algebraic curve) for which $f(p_i) = r_i$ at least $n - \tau$ values. Again, if $f(x)$ is the transmitted message polynomial, τ errors in the received R has been corrected.

3.5.3 The Guruswami-Sudan Algorithm

Guruswami and Sudan [7, 43] later improved their work to list decode of Reed-Solomon and algebraic-geometric codes with nearly all rate beyond boundary τ_{unique}, called the Guruswami-Sudan (GS) algorithm.

For Reed-Solomon codes, improvement is made based on defining (x_i, r_i) as a "singularity" of polynomial $Q(x, y)$. It means $Q(x, y)$ does not only pass point (x_i, r_i), but also intersects it by a number of times. The number of intersection is defined as *multiplicity m* (m is a positive integer). As mentioned in section 3.3.2, the general polynomial for GS decoding Reed-Solomon codes can be written as:

$$Q(x, y) = \sum_{a,b\in \mathrm{N}} Q_{ab} x^a y^b \tag{3.27}$$

where $Q_{ab} \in \mathrm{GF}(q)$. It can be seen that $Q(0, 0) = 0$ and $(0, 0)$ is a point at $Q(x, y)$. If there is no term $x^a y^b$ with total degree $a + b$ less than m as: $Q_{ab} = 0$ for $a + b < m$, $Q(x, y)$ has a zero of multiplicity m at $(0, 0)$. $(0, 0)$ is a singularity of polynomial $Q(x, y)$. Geometrically, $Q(x, y)$ intersects $(0, 0)$ m times. In general, to define point (x_i, r_i) as a singularity of a polynomial $Q(x, y)$ (3.27), $Q(x, y)$ shall be able to be written as:

$$Q^{(i)}(x, y) = \sum_{\alpha,\beta\in \mathrm{N}} Q^{(i)}_{\alpha\beta} (x - x_i)^{\alpha} (y - r_i)^{\beta} \tag{3.28}$$

where $Q^{(i)}_{\alpha\beta} \in \mathrm{GF}(q)$. It can be easily observed that (x_i, r_i) is a point of $Q^{(i)}(x, y)$ as $Q^{(i)}(x_i, r_i) = 0$. If $Q^{(i)}_{\alpha\beta} = 0$ for $\alpha + \beta < m$, $Q^{(i)}(x, y)$ has a zero of multiplicity m at (x_i, r_i) and (x_i, r_i) is a singularity of $Q^{(i)}(x, y)$. Therefore, for a general polynomial $Q(x, y)$, determine the relationship between its coefficient Q_{ab} and $Q^{(i)}(x, y)$'s coefficients $Q^{(i)}_{\alpha\beta}$ is critical to define point (x_i, r_i) as a singularity. This will be further demonstrated in Chapter 4 which presents GS decoding Reed-Solomon codes. Referring to section 3.3.2, $(x - x_i)^{\alpha}$ is the zero basis function with respect to x_i. Similarly, $(y - r_i)^{\beta}$ is also the zero basis function with respect to r_i. They are introduced in section 3.3.2 on the purpose of defining a point as a singularity of a polynomial.

Therefore, the first step of the GS algorithm can be described as: to construct a polynomial $Q(x, y)$ which has a zero of multiplicity at least m at units (x_0, r_0), (x_1, r_1), …, (x_{n-1}, r_{n-1}). This step is called interpolation. The second step of the GS algorithm is the same as described in section 3.5.2, called factorisation. Using the GS algorithm to decode a (n, k) Reed-Solomon code, the algorithm can correct up to [7]:

$$\tau_{\mathrm{GS}} = n - \left\lfloor \sqrt{n(n-d)} \right\rfloor - 1 \tag{3.29}$$

errors. As Hamming distance $d = n - k + 1$, τ_{GS} will be increased as code rate k/n decreases, indicating the GS algorithm has greater error-correction potential for low rate codes.

Similarly, for algebraic-geometric codes, polynomials operated in the GS algorithm can generally be written as:

$$Q(x, y, z) = \sum_{a,b\in \mathrm{N}} Q_{ab}\phi_a z^b \tag{3.30}$$

where ϕ_a is the rational function defined in $L(\infty p_\infty)$, where p_∞ is the point at infinity on the corresponding algebraic curve. To define interpolated unit (p_i, r_i) as a singularity of $Q(x, y, z)$, the zero basis functions with respect to affine point p_i and received word r_i are needed to be known. As described above, the zero basis function with respective to r_i can be written as: $(z - r_i)^\beta$. However, the zero basis functions with respect to the affine points on most of the algebraic curves are still unknown. In the literature, only the zero basis functions of the affine points on Hermitian curves have been developed [44], which is described in section 3.4.2. Hence, it is only feasible to apply the GS algorithm for Hermitian codes. For Hermitian codes, to define (p_i, r_i) as a singularity of $Q(x, y, z)$ (3.30), $Q(x, y, z)$ can be written as:

$$Q^{(i)}(x, y, z) = \sum_{\alpha,\beta\in \mathrm{N}} Q^{(i)}_{\alpha\beta}\psi_{p_i,a}(z - r_i)^\beta \tag{3.31}$$

Based on $\psi_{p_i,\alpha}$'s definition (3.25), it is not difficult to realise that $Q^{(i)}(p_i, r_i) = 0$. In (3.31), if $Q^{(i)}_{\alpha\beta} = 0$ for $\alpha + \beta < m$, then (p_i, r_i) is a singularity of $Q^{(i)}(x, y, z)$ and $Q^{(i)}(x, y, z)$ has a zero of multiplicity at least m at (p_i, r_i). The relationship between $Q(x, y, z)$'s coefficients Q_{ab} and $Q^{(i)}(x, y, z)$'s coefficients $Q^{(i)}_{\alpha\beta}$ is further demonstrated in Chapter 6 which presents GS decoding Hermitian codes.

Therefore, decoding Hermitian codes using the GS algorithm, interpolation is to find the polynomial $Q(x, y, z)$ which has a zero of multiplicity at least m at interpolation units: (p_0, r_0), (p_1, r_1), …, (p_{n-1}, r_{n-1}). Then factorisation is to find the transmitted message polynomial $f(x, y)$ defined in $L_w(l\ p_\infty)$ ($l = k + g - 1$ and p_∞ is a point at infinity on Hermitian curve $x^{w+1} + y^w + y = 0$). GS decoding Hermitian codes can correct errors up to:

$$\tau_{\mathrm{GS}} = n - \left\lfloor \sqrt{n(n - d^*)} \right\rfloor - 1 \tag{3.32}$$

where designed minimal distance $d^* = n - k - g + 1$. Again, indicated by (3.32), the GS algorithm has more error-correction potential for low rate codes.

3.5.4 The Koetter-Vardy Algorithm

In [7], it is demonstrated that further extension of the GS algorithm can be done by assigning a non-negative integer weight w_i to interpolated unit (x_i, r_i) such that $\sum_{i:p(x_i)=r_i} w_i > \sqrt{k \cdot \sum_{i=0}^{n-1} w_i^2}$. The extension idea gives two releases to the GS algorithm: first, the number of interpolated units can be greater than n. Second, with respect to different interpolated unit (x_i, r_i), different multiplicity value m_i can be assigned. This idea is later developed by Koetter and Vardy [8] who presented a soft-decision list decoding algorithm for Reed-Solomon codes, called the Koetter-Vardy (KV) algorithm.

In the KV algorithm, before interpolation and factorisation, an extra step that converts the received information's posteriori transition probability values to multiplicity values is performed. As a result of that, the number of interpolated units operated in interpolation is increased and each of them is assigned with a rational multiplicity value. It is shown that the KV's algorithm can easily perform beyond τ_{GS} (3.29) for Reed-Solomon codes. Details of decoding Reed-Solomon codes with the KV algorithm are presented in Chapter 5. Based on the KV algorithm, this project has also developed a soft-decision list decoding algorithm for Hermitian codes. It is also shown that soft-decision list decode Hermitian codes can easily perform beyond boundary τ_{GS} (3.32). Details of soft-decision list decoding of Hermitian codes are presented in Chapter 7.

3.6 Conclusions

The chapter presented the fundamental knowledge of algebraic-geometric codes, including algebraic curves, affine points on the curve and the rational functions associated with the curve. Based on this knowledge, a general description of encoding an algebraic-geometric code is given. Two classes of important algebraic-geometric codes are introduced in the chapter: Reed-Solomon codes and Hermitian codes. Their encoding processes are described with worked examples. To demonstrate the list decoding system, the pole basis and zero basis of these two kinds of codes are also introduced. At the end, a brief introduction to the list decoding system was given presenting chronological history of its development, from list decoding of low rate codes to list decoding of all rate codes and extending to soft-decision list decoding. It has been demonstrated that how list decoding can correct errors beyond the half distance boundary. This prerequisite knowledge lays a foundation to explain the list decoding of Reed-Solomon codes and Hermitian codes using hard decisions and soft decisions later in the thesis.

Chapter 4

Hard-Decision List Decoding of Reed-Solomon Codes

4.1 Introduction

This chapter presents a hard-decision list decoding algorithm for Reed-Solomon codes, the so-called the Guruswami-Sudan (GS) algorithm. The algorithm consists of interpolation and factorisation, which can be implemented by an iterative polynomial construction algorithm and a recursive coefficient search algorithm respectively. The algorithm's high decoding complexity is mainly dominated by the iterative interpolation process. Therefore, a novel complexity reduction modification scheme for the interpolation process has been developed in order to improve the algorithm's efficiency. The modification scheme is based on identifying any unnecessary polynomials during the iterative process and eliminating them. A worked example of this modification scheme is shown in the chapter for clarification. An algebraic-geometric explanation of the GS algorithm with the complexity reduction modification is presented with simulation results of Reed-Solomon codes for different list decoding parameters over the additive white Gaussian noise (AWGN) and Rayleigh fading channels. A complexity analysis is also shown comparing the GS algorithm with our modified GS algorithm, showing the modification can reduce complexity significantly in low error weight situations. This work is published in two papers by the author [52, 53].

4.2 Overview of the GS Algorithm

We first denote some commonly used symbols in this chapter:

• $F_q[x]$ – the ring of polynomials with coefficients from GF(q) and variable x, which can be generally written as: $f(x) = \sum_{a \in \mathrm{N}} f_a x^a$, $f_a \in$ GF(q).

• $F_q[x^w]$ – the subset of $F_q[x]$ with x degree $\leq w$

• $F_q[x, y]$ – the ring of bivariate polynomials with coefficients from GF(q) and variables x and y, which can be generally written as: $f(x, y) = \sum_{a,b \in \mathrm{N}} f_{ab} x^a y^b$, $f_{ab} \in$ GF(q).

The generation of a (n, k) Reed-Solomon code is defined by equations (3.13) and (3.14) in Chapter 3.

4.2.1 Interpolation and Factorisation

Interpolation: If the received word is $R = (r_0, r_1, ..., r_{n-1})$ ($r_i \in \mathrm{GF}(q)$, $i = 0, 1, \ldots, n - 1$), then combining with the finite field elements used in encoding $(x_0, x_1, ..., x_{n-1}) \in \mathrm{GF}(q)\backslash\{0\}$, n interpolated points can be formed as: (x_0, r_0), (x_1, r_1), …, (x_{n-1}, r_{n-1}). The task of interpolation is to construct a bivariate polynomial: $Q(x, y) = \sum_{a,b\in\mathrm{N}} Q_{ab}x^a y^b$

(3.27), which has a zero of multiplicity at least m over these n points and with minimal (1, k-1)-weighted degree which is explained later. $Q_{ab} \in \mathrm{GF}(q)$ is the coefficient of $x^a y^b$. Geometrically, this polynomial intersects the n points at least m times.

Factorisation: After the bivariate polynomial $Q(x, y)$ has been found, it is factorised in order to find the list L of polynomials $p(x)$ given by:

$$L = \{p(x)\text{: } (y - p(x)) \mid Q(x, y) \text{ and } \deg(p(x)) < k\} \tag{4.1}$$

All the polynomials in L have the possibility of being the transmitted message $f(x)$. The one with the minimum distance to the received word after re-encoding is chosen by the decoder.

4.2.2 Decoding Parameters

If we define the (u, v)-weighted degree of monomial $x^a y^b$ as:

$$\deg_{u,v}(x^a y^b) = au + bv \tag{4.2}$$

a sequence of bivariate monomials can be arranged by their weighted degrees. In order to decode a (n, k) Reed-Solomon code by the GS algorithm, the (1, k-1) - lexicographic order (ord) is used. Under (1, k-1) - lexicographic order [45, 54]:

$$x^{a_1} y^{b_1} < x^{a_2} y^{b_2}$$

if $\deg_{1,\,k\text{-}1}(x^{a_1} y^{b_1}) < \deg_{1,\,k\text{-}1}(x^{a_2} y^{b_2})$, or $\deg_{1,\,k\text{-}1}(x^{a_1} y^{b_1}) = \deg_{1,k\text{-}1}(x^{a_1} y^{b_1})$ and $a_1 > a_2$. For example, in order to decode a (7, 5) RS code, (1, 4) - lexicographic order is used. The generation of this order is shown by Table 4.1. The entries E_{ab} in Table 4.1a and 4.1b represent the (1, 4) - weighted degree and (1, 4) – lexicographic order of monomials M with x degree a and y degree b respectively. Applying (4.2) with $u = 1$ and $v = 4$, we can generate the (1, 4) - weighted degree of monomials M shown by Table 4.1a. Based on Table 4.1a and applying the above lexicographic order rule, we can generate the (1, 4) – lexicographic order of monomials M shown in Table 4.1b and denoted as ord(M). From Table 4.1b, it is easy to observe that $x^4 < x^2y < y^2$, since ord(x^4) = 4, ord(x^2y) = 9 and ord(y^2) = 14.

a b	0	1	2	3	4	5	6	7	8	9	10	11	12	…
0	0	1	2	3	4	5	6	7	8	9	10	11	12	…
1	4	5	6	7	8	9	10	11	12	…				
2	8	9	10	11	12	…								
3	12	…												
⋮	⋮													

Table 4.1a (1, 4) – weighted degree of monomial x^ay^b

a b	0	1	2	3	4	5	6	7	8	9	10	11	12	…
0	0	1	2	3	4	6	8	10	12	15	18	21	24	…
1	5	7	9	11	13	16	19	22	25	…				
2	14	17	20	23	26	…								
3	27	…												
⋮	⋮													

Table 4.1b (1, 4) – lexicographic order of monomial x^ay^b

Based on the monomial's weighted degree and order definition, we can define the weighted degree of a nonzero bivariate polynomial in $F_q[x, y]$ as the weighted degree of its leading monomial M_L. Any nonzero bivariate polynomial $Q(x, y)$ can be written as:

$$Q(x,y) = Q_0M_0 + Q_1M_1 + \cdots + Q_LM_L \tag{4.3}$$

with $M_0 < M_1 < \cdots < M_L$, $Q_0, Q_1, \cdots, Q_L \in$ GF(q) and $Q_L \neq 0$. The (1, k-1) - weighted degree of $Q(x, y)$ can be defined as:

$$\deg_{1,\,k\text{-}1}(Q(x, y)) = \deg_{1,\,k\text{-}1}(M_L) \tag{4.4}$$

L is called the leading order (lod) of polynomial $Q(x, y)$, defined as:

$$\text{lod}(Q(x, y)) = \text{ord}(M_L) = L \tag{4.5}$$

For example, give polynomial $Q(x, y) = 1 + x^2 + x^2y + y^2$, applying the above (1, 4)-lexicographic order, it has leading monomial $M_L = y^2$. Therefore, $\deg_{1,4}(Q(x, y)) = \deg_{1,4}(y^2) = 8$ and $\text{lod}(Q(x, y)) = \text{ord}(y^2) = 14$. Consequently, any two nonzero polynomials Q and H ($Q, H \in F_q[x, y]$) can be compared with respect to their leading order that:

$$Q \leq H, \text{ if } \text{lod}(Q) \leq \text{lod}(H) \tag{4.6}$$

$S_x(T)$ and $S_y(T)$ are denoted as the highest degree of x and y under the (1, k-1) - lexicographic order such that:

$$S_x(T) = \max\{a\text{: ord}(x^ay^0) \leq T\} \tag{4.7}$$

$$S_y(T) = \max\{b\text{: ord}(x^0y^b) \leq T\} \tag{4.8}$$

where T is any nonnegative integer. It is interesting to note that under (1, k-1) - lexicographic order x^ay^0 is the minimal monomial with weight degree a. Therefore, the (1, k-1) - weighted degree of any nonzero bivariate polynomial defined in (4.3) with leading order L can be determined as:

$$\deg_{1,\,k\text{-}1}(Q(x, y)) = S_x(L) \tag{4.9}$$

The error-correction capability t_m and the maximum number of candidate messages l_m in the output list with respect to a certain multiplicity m of the GS algorithm can be stated as [46]:

$$\tau_m = n - 1 - \left\lfloor \frac{S_x(C)}{m} \right\rfloor \tag{4.10}$$

$$l_m = S_y(C) \tag{4.11}$$

where

$$C = n\binom{m+1}{2} \tag{4.12}$$

C represents the number of iterations in the interpolation process. These parameters will be proven in section 4.4.1 when the factorisation theorem is presented. τ_m and l_m grow monotonically with multiplicity *m* [46]:

$$\tau_{m_1} \leq \tau_{m_2} \tag{4.13}$$

$$l_{m_1} < l_{m_2} \tag{4.14}$$

if $m_1 < m_2$. The GS algorithm algorithm's error-correction upper bound τ_{GS} for a (*n*, *k*) Reed-Solomon code is defined by equation (3.29). τ_{GS} is greater or equal to the half distance boundary τ defined by (3.15) and approaches to it asymptotically with code rate *k*/*n* increases. According to the GS algorithm analysis in [7], for Reed-Solomon codes, decoding capability of the GS algorithm merges with the conventional algebraic decoding algorithm at about k/n = 0.9. Note that the performance of the generalised minimum distance (GMD) decoding algorithm [55] does not depend on the code rate and it can always outperform the conventional decoding algorithm with marginal coding gains. Simulations results in [8] show that the GS algorithm can outperform the GMD algorithm in relatively low code rate situations. However, as code rate increases, the GS algorithm's performance will approach to the conventional decoding algorithm, and the GMD algorithm can slightly outperform the GS algorithm.

Now two examples are given to illustrate how τ_m and l_m grow with multiplicity *m* with the GS algorithm. Notice that those *m* listed in the following examples are the minimal values need to correct the corresponding number of errors τ_m.

Example 4.1: To decode Reed-Solomon code (63, 15) defined over GF(64), with code rate 0.238 (< 1/3), we obtain:

m	1	2	4	6	26
τ_m	27	30	31	32	33 = τ_{GS}
l_m	2	4	8	13	55

Example 4.2: To decode Reed-Solomon code (63, 31) defined over GF(64), with code rate 0.492 (> 1/3), we obtain:

m	1	3	5	13
τ_m	16	17	18	$19 = \tau_{GS}$
l_m	1	4	7	19

4.3. Interpolation

In this section, the interpolation theorem is explained from the algebraic-geometric point of view. This is followed by a detailed description of its implementation method and a novel modification scheme which improves its efficiency.

4.3.1. Interpolation Theorem

According to section 3.3.2, 1, x, …, x^a are the rational functions that have increasing pole orders [13] over the point of infinity p_∞ of a projective line. The interpolated polynomial can generally be written as: $Q(x, y) = \sum_{a,b\in \mathrm{N}} Q_{ab} x^a y^b$ (3.27).

1, (1 - x_i), …, $(1 - x_i)^\alpha$ are the rational functions that have increasing zero orders [13] over the finite field element x_i used in encoding, and the received word $r_i \in$ GF(q). The interpolated polynomial with respect to point (x_i, r_i) can also be written as:

$$Q(x, y) = \sum_{\alpha,\beta\in \mathrm{N}} Q_{\alpha\beta}^{(x_i,r_i)} (x - x_i)^\alpha (y - r_i)^\beta \qquad (4.15)$$

where $Q_{\alpha\beta}^{(x_i,r_i)} \in$ GF(q) is the coefficient of $(x - x_i)^\alpha (y - r_i)^\beta$. For (4.15), $Q(x_i, r_i) = 0$. Based on section 3.5.3, if $Q_{\alpha\beta}^{(x_i,r_i)} = 0$ for $\alpha + \beta < m$, $Q(x, y)$ has a zero of multiplicity at least m over (x_i, r_i).

It is important to notice that [46]:

$$x^a = (x - x_i + x_i)^a = \sum_{a\geq\alpha} \binom{a}{\alpha} x_i^{a-\alpha} (x - x_i)^\alpha \qquad (4.16)$$

and

$$y^b = (y - r_i + r_i)^b = \sum_{b \geq \beta} \binom{b}{\beta} r_i^{\,b-\beta} (y - r_i)^\beta \tag{4.17}$$

Substituting (4.16) and (4.17) into (3.27), we have:

$$Q(x, y) = \sum_{a,b \in \mathrm{N}} Q_{ab} \sum_{a \geq \alpha} \binom{a}{\alpha} x_i^{\,a-\alpha} (x - x_i)^\alpha \sum_{b \geq \beta} \binom{b}{\beta} r_i^{\,b-\beta} (y - r_i)^\beta$$

$$= \sum_{\alpha,\beta \in \mathrm{N}} \sum_{a \geq \alpha, b \geq \beta} Q_{ab} \binom{a}{\alpha} \binom{b}{\beta} x_i^{\,a-\alpha} r_i^{\,b-\beta} (x - x_i)^\alpha (y - r_i)^\beta \tag{4.18}$$

Therefore, from (4.15):

$$Q_{\alpha\beta}^{(x_i, r_i)} = \sum_{a \geq \alpha, b \geq \beta} Q_{ab} \binom{a}{\alpha} \binom{b}{\beta} x_i^{\,a-\alpha} r_i^{\,b-\beta} \tag{4.19}$$

This is the (α, β) - Hasse derivative evaluation on the point (x_i, r_i) of the polynomial $Q(x, y)$ defined by (3.27) [45, 56, 57]. (4.19) defines the constraints for the coefficients of polynomial Q (3.27) in order to have a zero of multiplicity m over point (x_i, r_i).

Example 4.3 Given polynomial $Q(x, y) = \sigma^5 + \sigma^5 x + y + xy$ defined in GF(8) in which σ is a primitive element satisfying $\sigma^3 + \sigma + 1 = 0$. Prove $Q(x, y)$ has a zero of multiplicity at least $m = 2$ at point (1, σ^5). Addition and multiplication table of GF(8) is shown in Appendix B.

Based on the above study, to have a zero of multiplicity 2 at (1, σ^5), we need $Q_{00}^{(1,\sigma^5)} = 0$, $Q_{01}^{(1,\sigma^5)} = 0$ and $Q_{10}^{(1,\sigma^5)} = 0$.

$$Q_{00}^{(1,\sigma^5)} = Q_{00} \binom{0}{0} \binom{0}{0} 1^{0\text{-}0} (\sigma^5)^{0\text{-}0} + Q_{10} \binom{1}{0} \binom{0}{0} 1^{1\text{-}0} (\sigma^5)^{0\text{-}0} + Q_{01} \binom{0}{0} \binom{1}{0} 1^{0\text{-}0} (\sigma^5)^{1\text{-}0} +$$
$$Q_{11} \binom{1}{0} \binom{1}{0} 1^{1\text{-}0} (\sigma^5)^{1\text{-}0} = 1 + 1 + \sigma^5 + \sigma^5 = 0$$

$$Q_{01}^{(1,\sigma^5)} = Q_{01} \binom{0}{0} \binom{1}{1} 1^{0\text{-}0} (\sigma^5)^{1\text{-}1} + Q_{11} \binom{1}{0} \binom{1}{1} 1^{1\text{-}0} (\sigma^5)^{1\text{-}1} = 1 + 1 = 0$$

$$Q_{10}^{(1,\sigma^5)} = Q_{10} \binom{1}{1} \binom{0}{0} 1^{1\text{-}1} (\sigma^5)^{0\text{-}0} + Q_{11} \binom{1}{1} \binom{1}{0} 1^{1\text{-}1} (\sigma^5)^{1\text{-}0} = \sigma^5 + \sigma^5 = 0.$$

Therefore, $Q(x, y)$ has a zero of multiplicity at least $m = 2$ at point $(1, \sigma^5)$.

If we use $D(Q)$ to denote the Hasse derivative evaluation of $Q(x, y)$, then (4.19) can be denoted as:

$$D_{\alpha\beta}Q(x_i, r_i) = \sum_{a\geq\alpha, b\geq\beta} Q_{ab}\binom{a}{\alpha}\binom{b}{\beta}x_i^{a-\alpha}r_i^{b-\beta} \tag{4.20}$$

Therefore, the interpolation of the GS algorithm can be generalised as: Find a minimal (1, k-1) - weighted degree polynomial $Q(x, y)$ that satisfies:

$$Q(x, y) = \min_{lod(Q)} \{Q(x, y) \in F_q[x, y] \mid D_{\alpha\beta}Q(x_i, r_i) = 0 \text{ for } i = 0, \ldots, n-1 \text{ and } \alpha + \beta < m \quad (\alpha, \beta \in \mathrm{N})\} \tag{4.21}$$

4.3.2. Iterative Polynomial Construction

To find interpolated polynomial (4.21), an iterative polynomial construction algorithm [9, 44-47, 58] is employed. In this algorithm, a group of polynomials are initialised, they are tested by each of the Hasse derivative evaluations (4.19) and modified interactively. The interactive modification between two polynomials is based on the following two properties of the Hasse derivative [45, 56].

Property 1: Linear Functional of Hasse derivative

If $H, Q \in F_q[x, y]$, d_1 and $d_2 \in \mathrm{GF}(q)$, then

$$D(d_1H + d_2Q) = d_1D(H) + d_2D(Q) \tag{4.22}$$

Property 2: Bilinear Hasse derivative

If $H, Q \in F_q[x, y]$, then

$$[H, Q]_D = HD(Q) - QD(H) \tag{4.23}$$

If the Hasse derivative evaluation of $D(Q) = d_1$ and $D(H) = d_2$ ($d_1, d_2 \neq 0$), based on Property 1 it is obvious to conclude that the Hasse derivative evaluation of (4.23) is zero, denoted as:

$$D([H, Q]_D) = 0 \tag{4.24}$$

If lod(H) > lod(Q), the new constructed polynomial from (4.23) has leading order lod(H). Therefore, by performing the bilinear Hasse derivative over two polynomials both of which have nonzero evaluations, we can reconstruct a polynomial which has zero Hasse derivative evaluation. Based on this principle, implementation algorithm for interpolation is to iteratively modify a set of polynomials through all n points and with every possible (α, β) pair under each point.

With multiplicity m, there are $\binom{m+1}{2}$ pairs of (α, β), which are arranged as: (α, β) = (0, 0), (0, 1), …, (0, m-1), (1, 0), (1, 1), …, (1, m-2), …, (m-1, 0). Therefore, when decoding a (n, k) Reed-Solomon code with multiplicity m, there are $C = n\binom{m+1}{2}$ iterations in order to construct a polynomial defined by (4.21). In order to introduce our complexity reduction modification to this interpolation algorithm, the iterative process is presented in a sequential manner with index i_k such that $i_k = i\binom{m+1}{2} + r$, where i denotes the index of points (i = 0, 1, …, n - 1), r denotes the index of (α, β) pairs ($r = 0, 1, \ldots, \binom{m+1}{2} - 1$).

At the beginning, a group of polynomials are initialised as

$$G_0 = \{Q_{0,j} = y^j, j = 0, 1, \ldots, l_m\} \tag{4.25}$$

where l_m is the maximal number of messages in the output list defined by (4.11). If M_L denotes the leading monomial of polynomial Q, it is important to point out that:

$$Q_{0,j} = \min\{Q(x, y) \in F_q[x, y] \mid \deg_y(M_L) = j\} \tag{4.26}$$

Under i_k modification, each polynomial in group G_{i_k} is tested by (4.20) using:

$$\Delta_j = D_{i_k}(Q_{i_k,j}) \tag{4.27}$$

Those polynomials with $\Delta_j = 0$ do not need to be modified. However, those polynomials with $\Delta_j \neq 0$ need to be modified based on (4.23). In order to construct a group of polynomials which satisfy:

$$Q_{i_k+1,j} = \min\{Q \in F_q[x, y] \mid D_{i_k}(Q_{i_k+1,j}) = 0, D_{i_k-1}(Q_{i_k+1,j}) = 0, \cdots, D_0(Q_{i_k+1,j}) = 0 \text{, and } \deg_y(M_L) = j\} \quad (4.28)$$

the minimal polynomial among those polynomials with $\Delta_j \neq 0$ is chosen. Denote its index as $j^{'}$ and record it as $Q^{'}$:

$$j^{'} = \text{index}(\min\{Q_{i_k,j} \mid \Delta_j \neq 0\}) \quad (4.29)$$

$$Q^{'} = Q_{i_k,j^{'}} \quad (4.30)$$

For those polynomials with $\Delta_j \neq 0$ but $j \neq j^{'}$, modify them by (4.23) without the leading order increasing:

$$Q_{i_k+1,j} = [Q_{i_k,j}, Q^{'}]_{D_{i_k}} = \Delta_{j^{'}} Q_{i_k,j} - \Delta_j Q^{'} \quad (4.31)$$

Based on (4.24), we know that $D_{i_k}(Q_{i_k+1,j}) = 0$. As lod($Q_{i_k,j}$) > lod($Q^{'}$), therefore lod($Q_{i_k+1,j}$) = lod($Q_{i_k,j}$).

For $Q^{'}$ itself, it is modified by (4.23) with the leading order increasing:

$$Q_{i_k+1,j^{'}} = [xQ^{'}, Q^{'}]_{D_{i_k}} = \Delta_{j^*}(x - x_i)\, Q^{'} \quad (4.32)$$

where x_i is the x – coordinate of current interpolating point (x_i, r_i). $\Delta_{j^*} = D_{i_k}(Q^{'}) \neq 0$ and so as $D_{i_k}(xQ^{'}) \neq 0$, therefore, $D_{i_k}(Q_{i_k+1,j^{'}}) = 0$. As lod($xQ^{'}$) > lod($Q^{'}$), lod($Q_{i_k+1,j^{'}}$) = lod($xQ^{'}$) > lod($Q_{i_k,j^{'}}$). Therefore whenever (4.32) is performed, we have: lod($Q_{i_k+1,j}$) > lod($Q_{i_k,j}$).

After C iterative modifications, the minimal polynomial in group G_c is the interpolated polynomial that satisfies (4.21), and it is chosen to be factorised in the next step:

$$Q(x, y) = \min\{Q_{C,j} \mid Q_{C,j} \in G_C\} \quad (4.33)$$

4.3.3. Complexity Reduced Modification

Based on the above analysis, it can be observed that when decoding a (n, k) Reed-Solomon code with multiplicity m, $l_m + 1$ bivariate polynomials are being interactively modified over C iterative steps in which Hasse derivative evaluation (4.20) and bilinear Hasse derivative modification (4.23) are being performed. This process has complexity approximately $O(n^2m^4)$ [46] and is responsible for the GS algorithm's high decoding complexity. Therefore, reducing the complexity of interpolation is essential to improve the algorithm's efficiency.

The leading order of the polynomial group G_{i_k} is defined as the minimal leading order among the group's polynomials:

$$\text{lod}(G_{i_k}) = \min\{\text{lod}(Q_{i_k,j}) \mid Q_{i_k,j} \in G_{i_k}\} \tag{4.34}$$

Based on initialisation defined in (4.25), the leading order of polynomial group G_0 is $\text{lod}(G_0) = \text{lod}(Q_{0,0}) = 0$. In the i_k modification, if no polynomial needs to be modified, then the polynomial group is unchanged, $\text{lod}(G_{i_k+1}) = \text{lod}(G_{i_k})$. Once a polynomial needs to be modified, (4.32) must be used. If M_L is the leading monomial of Q^*, we have:

$$\text{lod}(xQ^*) = \text{lod}(Q^*) + \left\lfloor \frac{\deg_x Q^*}{k-1} \right\rfloor + \deg_y(M_L) + 1 \tag{4.35}$$

and $\text{lod}(G_{i_k})$ will be increased if Q^* is the minimal polynomial in the group G_{i_k}. The leading order increase guarantees that in the i_k iterative step, the leading order of the polynomials group G_{i_k} is always less than or equal to i_k:

$$\text{lod}(G_{i_k}) \le i_k \tag{4.36}$$

Based on (4.36), after C iterative steps, we have:

$$\text{lod}(G_C) \le C \tag{4.37}$$

From (4.33) we know that only the minimal polynomial is chosen from the polynomial group G_C as $Q(x, y) = \{Q_{c,j} \mid Q_{c,j} \in G_c \text{ and } \text{lod}(Q_{c,j}) = \text{lod}(G_c)\}$, therefore:

$$\text{lod}(Q(x, y)) \le C \tag{4.38}$$

which means the interpolated polynomial $Q(x, y)$ has leading order less than or equal to C. Those polynomials with leading order over C will not be candidates to be $Q(x, y)$. Therefore, during the iterative process, we can modify the group of polynomials by eliminating those with leading order over C as [52]:

$$G_{i_k} = \{ Q_{i_k,j} \mid \text{lod}(Q_{i_k,j}) \le C\} \tag{4.39}$$

We now prove this modification will not affect the final result. In i_k iterative step, if there is a polynomial $Q_{i_k,j}$ with lod($Q_{i_k,j}$) > C, it may be modified either by (4.31) or (4.32) which will result in its leading order being unchanged or increased. Therefore, at the end lod($Q_{c,j}$) > C and based on (4.38) it can not be $Q(x, y)$. However, if $Q_{i_k,j}$ is the minimal polynomial defined by (4.29), this implies that those polynomials with leading order less than C do not need to be modified. If $Q_{i_k,j}$ is not the minimal polynomial defined by (4.29), $Q_{i_k,j}$ will not be chosen to perform bilinear Hasse derivative (4.31) with other polynomials. Therefore, $Q(x, y)$ has no information introduced from $Q_{i_k,j}$ since lod($Q_{i_k,j}$) > C. As a result, eliminating the polynomials with leading order over C will not affect the final outcome.

This complexity modification scheme can be generally applied to the iterative interpolation process, such as soft-decision list decoding of Reed-Solomon codes and hard/soft-decision list decoding of Hermitian codes, both of which will be presented in the later chapters of this thesis. Based on the total number of iterations C for interpolation, the interpolated polynomial's leading order always satisfies: lod($Q(x, y)$) $\le C$. It implies that those polynomials in the group G can be eliminated once their leading order is over C.

This modification can reduce some unnecessary computation in terms of avoiding Hasse derivative evaluation (4.27) and bilinear Hasse derivative modification (4.31) (4.32) of polynomials with leading order over C. Based on the above analysis, the modified interpolation process can be summarised as:

Algorithm 4.1: Interpolation for list decoding of a (*n*, *k*) Reed-Solomon code

(i) Initialise a group of polynomials by (4.25), set $i_k = 0$

(ii) Modify the polynomial group by (4.39)

(iii) Perform Hasse derivative evaluation (4.27) for each polynomial in the group

(iv) If all the polynomials' Hasse derivative evaluations are zero, go to (vii)

(v) Find the minimal polynomial defined by (4.29) (4.30)

(vi) For the minimal polynomial, modify it by (4.32). For the other polynomials with nonzero Hasse derivative evaluation, modify them by (4.31)

(vii) $i_k = i_k + 1$

(viii) If $i_k = C$, stop the process and choose $Q(x, y)$ defined by (4.33) else go to (ii).

Here an example is given showing the modified interpolation process.

Example 4.4: Decode the (7, 2) Reed-Solomon code defined over GF(8) with multiplicity $m = 2$. As $C = 7\binom{3}{1} = 21$, based on (4.10) (4.11) we have $\tau_2 = 3$ and $l_2 = 5$. The transmitted codeword is generated by evaluating the message polynomial $f(x) = \sigma + \sigma^6 x$ over the set of points $x = (1, \sigma, \sigma^3, \sigma^2, \sigma^6, \sigma^4, \sigma^5)$ and the corresponding received word is $R = (\sigma^5, \sigma^3, \sigma^4, 0, \sigma^6, \sigma^2, \sigma^2)$, where σ is a primitive element in GF(8) satisfying $\sigma^3 + \sigma + 1 = 0$. Construct a bivariate polynomial that has a zero of multiplicity $m = 2$ over the n points $(x_i, r_i)|_{i=0}^{n-1}$.

At the beginning, 6 polynomials are initialised as:

$Q_{0,0} = 1$, $Q_{0,1} = y$, $Q_{0,2} = y^2$, $Q_{0,3} = y^3$, $Q_{0,4} = y^4$, and $Q_{0,5} = y^5$. Their leading orders are $\text{lod}(Q_{0,0}) = 0$, $\text{lod}(Q_{0,1}) = 2$, $\text{lod}(Q_{0,2}) = 5$, $\text{lod}(Q_{0,3}) = 9$, $\text{lod}(Q_{0,4}) = 14$ and $\text{lod}(Q_{0,5}) = 20$ respectively. $\text{lod}(G_0) = \text{lod}(Q_{0,0}) = 0$.

When $i_k = 0$, $i = 0$ and $(\alpha, \beta) = (0, 0)$, no polynomial is eliminated from the group G_0. Perform Hasse derivative evaluation for each of the polynomials in G_0 as:

$\Delta_0 = D_{i_k}(Q_{i_k,0}) = D_{(0,0)}^{(x_0,y_0)}(Q_{0,0}) = 1$, $\Delta_1 = D_{i_k}(Q_{i_k,1}) = D_{(0,0)}^{(x_0,y_0)}(Q_{0,1}) = \sigma^5$

$\Delta_2 = D_{i_k}(Q_{i_k,2}) = D_{(0,0)}^{(x_0,y_0)}(Q_{0,2}) = \sigma^3$, $\Delta_3 = D_{i_k}(Q_{i_k,3}) = D_{(0,0)}^{(x_0,y_0)}(Q_{0,3}) = \sigma$

$\Delta_4 = D_{i_k}(Q_{i_k,4}) = D_{(0,0)}^{(x_0,y_0)}(Q_{0,4}) = \sigma^7$, $\Delta_5 = D_{i_k}(Q_{i_k,5}) = D_{(0,0)}^{(x_0,y_0)}(Q_{0,5}) = \sigma^4$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$j' = 0$ and $Q' = Q_{0,0}$

Modify polynomials in G_0 with $\Delta_j \neq 0$ as:

$Q_{1,0} = \Delta_0(x - x_0)Q' = 1 + x$, and $\text{lod}(Q_{1,0}) = 1$

$Q_{1,1} = \Delta_0 Q_{0,1} - \Delta_1 Q' = \sigma^5 + y$, and $\text{lod}(Q_{1,1}) = 2$

$Q_{1,2} = \Delta_0 Q_{0,2} - \Delta_2 Q' = \sigma^3 + y^2$, and $\text{lod}(Q_{1,2}) = 5$

$Q_{1,3} = \Delta_0 Q_{0,3} - \Delta_3 Q' = \sigma + y^3$, and $\text{lod}(Q_{1,3}) = 9$

$Q_{1,4} = \Delta_0 Q_{0,4} - \Delta_4 Q' = \sigma^6 + y^4$, and $\text{lod}(Q_{1,4}) = 14$

$Q_{1,5} = \Delta_0 Q_{0,5} - \Delta_5 Q' = \sigma^4 + y^5$, and $\text{lod}(Q_{1,5}) = 20$

$\text{lod}(G_1) = \text{lod}(Q_{1,0}) = 1$.

When $i_k = 1$, $i = 0$ and $(\alpha, \beta) = (0, 1)$, no polynomial is eliminated from the group G_1.

Perform Hasse derivative evaluation for each of the polynomial in G_1 as:

$\Delta_0 = D_{i_k}(Q_{i_k,0}) = D_{(0,1)}^{(x_0,y_0)}(Q_{1,0}) = 0$. $\Delta_1 = D_{i_k}(Q_{i_k,1}) = D_{(0,1)}^{(x_0,y_0)}(Q_{1,1}) = 1$

$\Delta_2 = D_{i_k}(Q_{i_k,2}) = D_{(0,1)}^{(x_0,y_0)}(Q_{1,2}) = 0$. $\Delta_3 = D_{i_k}(Q_{i_k,3}) = D_{(0,1)}^{(x_0,y_0)}(Q_{1,3}) = \sigma^3$

$\Delta_4 = D_{i_k}(Q_{i_k,4}) = D_{(0,1)}^{(x_0,y_0)}(Q_{1,4}) = 0$. $\Delta_5 = D_{i_k}(Q_{i_k,5}) = D_{(0,1)}^{(x_0,y_0)}(Q_{1,5}) = \sigma^6$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$j' = 1$ and $Q' = Q_{1,1}$

As $\Delta_0 = \Delta_2 = \Delta_4 = 0$,

$Q_{2,0} = Q_{1,0} = 1 + x$, and $\text{lod}(Q_{2,0}) = 1$

$Q_{2,2} = Q_{1,2} = \sigma^3 + y^2$, and $\text{lod}(Q_{2,2}) = 5$

$Q_{2,4} = Q_{1,4} = \sigma^6 + y^4$, and $\text{lod}(Q_{2,4}) = 14$

Modify polynomials in G_1 with $\Delta_j \neq 0$ as:

$Q_{2,1} = \Delta_1(x - x_0)Q' = \sigma^5 + \sigma^5 x + y(1 + x)$, $\mathrm{lod}(Q_{2,1}) = 4$

$Q_{2,3} = \Delta_1 Q_{1,3}$ - $\Delta_3 Q' = \sigma^3 y + y^3$, $\mathrm{lod}(Q_{2,3}) = 9$

$Q_{2,5} = \Delta_1 Q_{2,5}$ - $\Delta_5 Q' = \sigma^6 y + y^5$, $\mathrm{lod}(Q_{2,5}) = 20$

$\mathrm{lod}(G_2) = \mathrm{lod}(Q_{2,0}) = 1$.

When $i_k = 2$, $i = 0$ and $(\alpha, \beta) = (1, 0)$, no polynomial is eliminated from the group G_2.

Perform Hasse derivative evaluation for each of the polynomial in G_2 as:

$\Delta_0 = D_{i_k}(Q_{i_k,0}) = D_{(1,0)}^{(x_0,y_0)}(Q_{2,0}) = 1$. $\Delta_1 = D_{i_k}(Q_{i_k,1}) = D_{(1,0)}^{(x_0,y_0)}(Q_{2,1}) = 0$

$\Delta_2 = D_{i_k}(Q_{i_k,2}) = D_{(1,0)}^{(x_0,y_0)}(Q_{2,2}) = 0$. $\Delta_3 = D_{i_k}(Q_{i_k,3}) = D_{(1,0)}^{(x_0,y_0)}(Q_{2,3}) = 0$

$\Delta_4 = D_{i_k}(Q_{i_k,4}) = D_{(1,0)}^{(x_0,y_0)}(Q_{2,4}) = 0$. $\Delta_5 = D_{i_k}(Q_{i_k,5}) = D_{(1,0)}^{(x_0,y_0)}(Q_{2,5}) = 0$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$j' = 0$ and $Q' = Q_{2,0}$

As $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = 0$

$Q_{3,1} = Q_{2,1} = \sigma^5 + \sigma^5 x + y(1 + x)$, $\mathrm{lod}(Q_{3,1}) = 4$

$Q_{3,2} = Q_{2,2} = \sigma^3 + y^2$, and $\mathrm{lod}(Q_{3,2}) = 5$

$Q_{3,3} = Q_{2,3} = \sigma^3 y + y^3$, $\mathrm{lod}(Q_{3,3}) = 9$

$Q_{3,4} = Q_{2,4} = \sigma^6 + y^4$, and $\mathrm{lod}(Q_{3,4}) = 14$

$Q_{3,5} = Q_{2,5} = \sigma^6 y + y^5$, $\mathrm{lod}(Q_{3,5}) = 20$

Modify polynomials in G_2 with $\Delta_j \neq 0$ as:

$Q_{3,0} = \Delta_0(x - x_0)Q' = 1 + x^2$, $\mathrm{lod}(Q_{3,0}) = 3$

$\mathrm{lod}(G_3) = \mathrm{lod}(Q_{3,0}) = 3$.

Based on the same process, interpolation is run through all the rest of the points (x_i, r_i) ($i = 1$ to 6). In order to illustrate the complexity reduction modification scheme, Table

i_k	0	1	2	3	4	5	6	7	8	9
lod($Q_{i_k,0}$)	0	1	1	3	6	6	10	15	15	21
lod($Q_{i_k,1}$)	2	2	4	4	4	7	7	7	11	11
lod($Q_{i_k,2}$)	5	5	5	5	5	5	5	5	5	5
lod($Q_{i_k,3}$)	9	9	9	9	9	9	9	9	9	9
lod($Q_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14
lod($Q_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20
lod(G_{i_k})	0	1	1	3	4	5	5	5	5	5

Original GS

i_k	10	11	12	13	14	15	16	17	18	19	20	21
lod($Q_{i_k,0}$)	28	28	36	45	45	55	55	55	55	66	66	78
lod($Q_{i_k,1}$)	11	16	16	16	22	22	22	22	22	22	29	29
lod($Q_{i_k,2}$)	5	5	5	5	5	5	8	8	12	12	12	12
lod($Q_{i_k,3}$)	9	9	9	9	9	9	9	13	13	13	13	13
lod($Q_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14	14	14
lod($Q_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20	20	20
lod(G_{i_k})	5	5	5	5	5	5	8	8	12	12	12	12

Modified GS

i_k	10	11	12	13	14	15	16	17	18	19	20	21
lod($Q_{i_k,0}$)	—	—	—	—	—	—	—	—	—	—	—	—
lod($Q_{i_k,1}$)	11	16	16	16	—	—	—	—	—	—	—	—
lod($Q_{i_k,2}$)	5	5	5	5	5	5	8	8	12	12	12	12
lod($Q_{i_k,3}$)	9	9	9	9	9	9	9	13	13	13	13	13
lod($Q_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14	14	14
lod($Q_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20	20	20
lod(G_{i_k})	5	5	5	5	5	5	8	8	12	12	12	12

Note: — means the corresponding polynomial is eliminated.

▨ means the corresponding polynomial is chosen as $Q(x, y)$.

Table 4.2 Iterative process of example 4.4

4.2 shows the whole iterative process with respect to the polynomials' leading order. From Table 4.2 we can see that the modified algorithm starts to take action at $i_k = 10$ when there is polynomial with leading order over 21 and eliminating those polynomials will not affect the final outcome. At the end, both the original and modified GS algorithm produce the same result: $Q(x, y) = \min\{G_{21}\} = Q_{21,2} = 1 + \sigma^4x^2 + \sigma^2x^4 + y^2(\sigma^5 + \sigma^4x^2)$. From this example we can see that more computation can be reduced if the modified algorithm starts to take action at earlier steps. A detailed complexity analysis of this modified algorithm is presented in section 4.5.

4.4 Factorisation

In this section, the factorisation theorem is explained, which is followed by a detailed description of its implementation method: the Roth-Ruckenstein's algorithm.

4.4.1 Factorisation Theorem

As mentioned in section 4.2.2, given the interpolated polynomial $Q(x, y)$, the transmitted message polynomial $f(x)$ can be found out by determining $Q(x, y)$'s y roots.

Lemma 4.1 If $Q(x, y)$ has a zero of multiplicity at least m over (x_i, r_i) and $p(x)$ is a polynomial in $F_q[x^{k-1}]$ that $p(x_i) = r_i$, then $(x - x_i)^m \mid Q(x, p(x))$ [7].

Define $\Lambda(p, R)$ as the number of symbols in received word R that satisfy $p(x_i) = r_i$ as:

$$\Lambda(p, R) = |\{i: p(x_i) = r_i, i = 0, 1, \ldots, n - 1\}| \tag{4.40}$$

Lemma 4.2 $p(x)$ is a polynomial in $F_q[x^{k-1}]$ and $p(x_i) = r_i$ for at least $\Lambda(p, R)$ values, if $m\,\Lambda(p, R) > \deg_{1,k-1}(Q(x, y))$, then $y - p(x) \mid Q(x, y)$, or $Q(x, p(x)) = 0$ [7].

Based on lemma 4.1, if $p(x_i) = r_i$, then $(x - x_i)^m \mid Q(x, y)$. If S is the set of i that satisfies $p(x_i) = r_i$. as $|S| = \Lambda(p, R)$, then $\prod_{i\in S}(x - x_i)^m \mid Q(x, p(x))$. Assume $g_1(x) = \prod_{i\in S}(x - x_i)^m$ and $g_2(x) = Q(x, p(x))$, therefore $g_1(x) \mid g_2(x)$. It is obvious that $g_1(x)$ has x degree $m\,\Lambda(p, R)$ and $g_2(x)$ has x degree equals to $\deg_{1,k\text{-}1}Q(x, y)$. If $m\,\Lambda(p, R) > \deg_{1,k\text{-}1}Q(x, y)$ and $g_1(x) \mid g_2(x)$, the only solution for these two preconditions is: $g_2(x) = 0$. Therefore, if $m\,\Lambda(p, R) > \deg_{1,k\text{-}1}(Q(x, y))$, $Q(x, p(x)) = 0$ or equivalently, $y - p(x) \mid Q(x, y)$.

As $Q(x, y)$ is the interpolated polynomial from the last step, according to (4.38), lod(Q) $\leq C$. Based on (4.9), $\deg_{1,k\text{-}1}(Q(x, y)) \leq S_x(C)$. If $m\,\Lambda(f, R) \geq S_x(C)$, then $m\,\Lambda(f, R) \geq \deg_{1,k\text{-}1}(Q(x, y))$. Based on lemma 4.2, if $\Lambda(f, R) \geq 1 + \left\lfloor \frac{S_x(C)}{m} \right\rfloor$, the transmitted message polynomial $f(x)$ can be found out by factorising $Q(x, y)$. As $\Lambda(f, R)$ represents the number of points that satisfy $r_i = f(x_i) = c_i$, those points that do not satisfy this equation are where the errors locate. Therefore the error-correction capability of the GS algorithm is $\tau_m = n - \left\lfloor \frac{S_x(C)}{m} \right\rfloor - 1$ which is defined by (4.10). Under (1, k-1)-lexicographic order, x^0y^j is the maximal monomial with weighted degree $(k - 1)j$. In polynomial $Q(x, y)$, there should not be any monomials with y-degree over $S_y(C)$, otherwise lod(Q) $> C$. As a result, $\max\{\deg_y Q(x, y)\} \leq S_y(C)$. As the factorisation output list contains the y roots of $Q(x, y)$, and the number of y roots of $Q(x, y)$ should not exceed its y degree, therefore the maximal number of candidate messages in the output list is $l_m = S_y(C)$ which is defined by (4.11).

4.4.2 Recursive Coefficient Search

To find out the y-roots of the interpolated polynomial $Q(x, y)$, Roth and Ruckenstein [10] introduced an efficient algorithm for factorising these bivariate polynomials, called Roth-Ruckenstein's algorithm.

In general, factorisation output $p(x) \in F_q[x^{k\text{-}1}]$ can be expressed in the form of:

$$p(x) = p_0 + p_1x + \cdots + p_{k\text{-}1}x^{k\text{-}1} \tag{4.41}$$

where $p_0, p_1, \ldots, p_{k-1} \in \mathrm{GF}(q)$. In order to find the polynomials $p(x)$, we must determine their coefficients $p_0, p_1, \ldots, p_{k-1}$ respectively. The idea of Roth-Ruckenstein's algorithm is to recursively deduce $p_0, p_1, \ldots, p_{k-1}$ one at a time.

For any bivariate polynomial, if h is the highest degree such that $x^h \mid Q(x, y)$, we can define:

$$Q^*(x,y) = \frac{Q(x,y)}{x^h} \tag{4.42}$$

If we denote $p_0 = p(x)$ and $Q_0(x, y) = Q^*(x, y)$, where $Q(x, y)$ is the new interpolated polynomial (4.33), we can define the recursive updated polynomials $p_s(x)$ and $Q_s(x, y)$, where $s \geq 1$, as:

$$p_s(x) = \frac{p_{s-1}(x) - p_{s-1}(0)}{x} = p_s + \cdots + p_{k-1}x^{k-1-s}, \; (s \leq k - 1) \tag{4.43}$$

$$Q_s(x,y) = Q^*_{s-1}(x, xy + p_{s-1}) \tag{4.44}$$

Lemma 4.3 In this sequential deduction with $p_s(x)$ and $Q_s(x, y)$ defined by (4.43) and (4.44), when $s \geq 1$, $(y - p(x)) \mid Q(x, y)$ if and only if $(y - p_s(x)) \mid Q_s(x, y)$ [46].

This means that if polynomial $p_s(x)$ is a y root of $Q_s(x, y)$, we can trace back to find the coefficients $p_{s-1}, \ldots, p_1, p_0$ to reconstruct the polynomial $p(x)$, which is the y root of polynomial $Q(x, y)$.

The first coefficient p_0 can be determined by finding the roots of $Q_0(0, y) = 0$. If we assume that $Q(x, p(x)) = 0$, then based on lemma 4.3, $p_0(x)$ should satisfy $Q_0(x, p_0(x)) = 0$. When $x = 0$, $Q_0(0, p_0(0)) = 0$. According to (4.41) $p_0(0) = p_0$, therefore p_0 is the root of $Q_0(0, y) = 0$. By finding the roots of $Q_0(0, y) = 0$, a number of different p_0 can be determined. For each p_0, we can deduce further to find the rest of p_s ($s = 1, \ldots, k - 1$) based on the recursive transformation (4.43) and (4.44).

Assume that after s - 1 deductions, polynomial $p_{s-1}(x)$ is the y root of $Q_{s-1}(x, y)$. Based on (4.43), $p_{s-1}(0) = p_{s-1}$ and a number of p_{s-1} can be determined by finding the roots of $Q_{s-1}(0, y) = 0$. For each p_{s-1}, we can deduce to find p_s. As $Q_{s-1}(x, p_{s-1}(x)) = 0$, $(y - p_{s-1}(x)) \mid Q_{s-1}(x, y)$. If we define $y = xy + p_{s-1}$, then $(xy + p_{s-1} - p_{s-1}(x)) \mid Q_{s-1}(x, xy + p_{s-1})$. Based on (4.43), $xy + p_{s-1} - p_{s-1}(x) = xy - xp_s(x)$. As $Q_s(x, y) = Q^*_{s-1}(x, xy + p_{s-1})$, $(xy - xp_s(x)) \mid Q_{s-1}(x, xy + p_{s-1})$, and $(y - p_s(x)) \mid Q_s(x, y)$. Therefore, p_s can again be determined by finding the roots of $Q_s(0, y) = 0$. This root finding algorithm can be explained as a tree growing process, which is shown in Fig 4.1. There can be an exponential number of routes for choosing coefficients p_s (s = 0, 1, ..., k - 1) to construct $p(x)$. However, the intended $p(x)$ should satisfy: $\deg(p(x)) < k$ and $(y - p(x)) \mid Q(x, y)$. Based on (4.43), when $s = k$, $p_k(x) = 0$. Therefore if $Q_k(x, 0) = 0$, or equivalently $Q_k(x, p_k(x)) = 0$, $(y-p_k(x)) \mid Q_k(x, y)$. According to lemma 4.3, $(y-p(x)) \mid Q(x, y)$ and $p(x)$ is found.

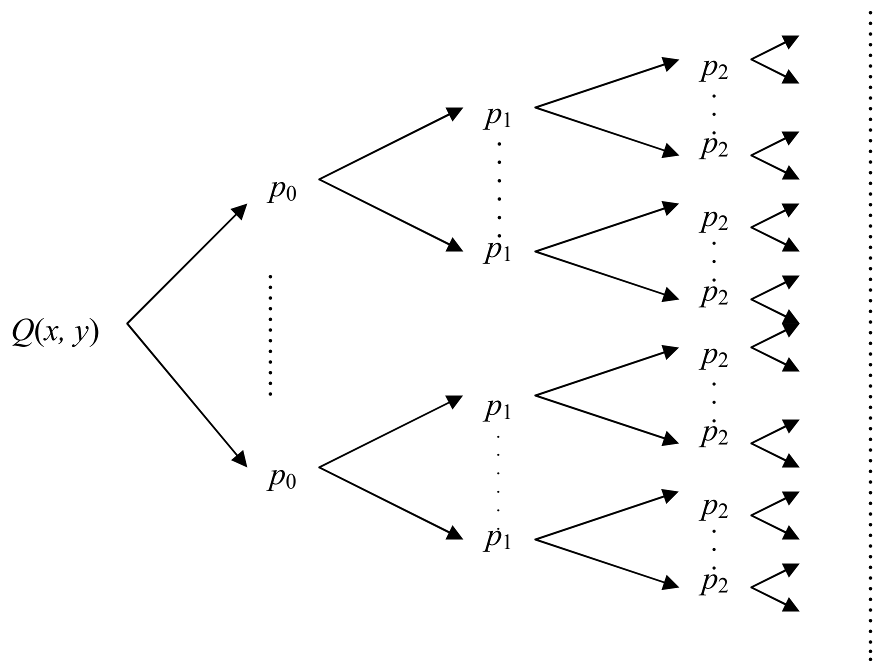


Figure 4.1 Coefficients deduction in the Roth-Ruckenstein's algorithm

Based on the above analysis, the factorisation process can be summarised as [10, 46]:

Algorithm 4.2: Factorisation of list decoding of a (n, k) Reed-Solomon code

(i) Initialise $Q_0(x, y) = Q^*(x, y)$, $s = 0$

(ii) Find roots p_s of $Q_s(0, y) = 0$

(iii) For each p_s, perform Q transformation (4.44) to calculate $Q_{s+1}(x, y)$

(iv) $s = s + 1$

(v) If $s < k$, go to (ii). If $s = k$ and $Q_s(x, 0) \neq 0$, stop this deduction route. If $s = k$ and $Q_s(x, 0) = 0$, trace the deduction route to find $p_{s-1}, \ldots, p_1, p_0$.

Here presents a work example of the Roth-Ruckenstein's algorithm.

Example 4.5: Based on polynomial $Q(x, y) = 1 + \sigma^4x^2 + \sigma^2x^4 + y^2(\sigma^5 + \sigma^4x^2)$ which is the interpolation result of example 4.4, determine the factorisation output list L using the Roth-Ruckenstein's algorithm.

Initialise $Q_0(x, y) = Q^*(x, y) = 1 + \sigma^4x^2 + \sigma^2x^4 + y^2(\sigma^5 + \sigma^4x^2)$ and $s = 0$.

$Q_0(0, y) = 1 + \sigma^5y^2$ and $p_0 = \sigma$ is the root of $Q_0(0, y) = 0$.

For $p_0 = \sigma$, generate $Q_1(x, y) = Q_0^*(x, xy + p_0) = \sigma^3 + \sigma^2x^2 + y^2(\sigma^5 + \sigma^4x^2)$. $s = s + 1 = 1$. As $s < k$, go to (ii) of the algorithm

$Q_1(0, y) = \sigma^3 + \sigma^5y^2$ and $p_1 = \sigma^6$ is a root of $Q_1(0, y) = 0$.

For $p_1 = \sigma^6$, generate $Q_2(x, y) = Q_1^*(x, xy + p_1) = y^2(\sigma^5 + \sigma^4x^2)$. $s = s + 1 = 2$. As $s = k$ and $Q_2(x, 0) = 0$, trace this deduction route to find its output $p_0 = \sigma$ and $p_1 = \sigma^6$.

As a result, factorisation output list $L = \{p(x) = \sigma + \sigma^6x\}$. According to example 4.4, $p(x)$ matches the transmitted message polynomial $f(x)$.

4.5 Complexity Analysis

The GS algorithm's high decoding complexity is mainly caused by interpolation. Compared to this, the factorisation complexity cost is insignificant. This section analyses the computational complexity (finite field arithmetic operations) for the original and modified algorithm.

It is difficult to analyse the computational complexity precisely because the length (number of coefficients) of the group of interpolated polynomials varies in different situations. We define the number of coefficients of the polynomial $Q(x, y)$ as its interpolation cost by:

$$\gamma = |\{Q_{ab} = \text{coeff}(Q(x, y)) \text{ and } Q_{ab} \neq 0\}| \tag{4.45}$$

[59] has stated that the interpolation cost is error dependent:

$$\gamma(e) \leq \frac{\Omega^2}{2(k-1)} + \frac{\Omega}{2} + \frac{\Phi(k-\Phi-1)}{2(k-1)} + m + 1 \tag{4.46}$$

where $\Omega = em + (k\text{-}1)m$, $\Phi = \Omega \bmod (k\text{-}1)$ and e is the error weight. According to section 4.3.3, we know that the interpolated polynomial $Q(x, y)$ has leading order less than or equal to C, therefore its interpolation cost is less than or equal to $C + 1$. If we regard the interpolation process as solving a system of homogeneous linear equations by Gaussian elimination and assume those polynomials have the same interpolation cost as $C + 1$, the GS algorithm's computational complexity can be predicted. Interpolating the group of polynomials with interpolation cost $C + 1$ by C iterative steps can be regarded as operating on a matrix of size $C \times (C + 1)$ by Gaussian elimination and its computational complexity is approximately [59]:

$$\frac{2}{3}(C+1)^3 \tag{4.47}$$

However, in most of the situations $\gamma(e) \leq C + 1$, which means some elements in the row of the matrix are not used and the row operation is not fully performed. Therefore, in most cases (4.47) is an upper bound for the GS algorithm's computational complexity. As the interpolation cost grows with the error weight, so does the computational complexity. Based on (4.47), Table 4.3 predicts the computational complexity for decoding Reed-Solomon codes (63, 15) and (63, 31) both of which were first introduced in example 4.1 and 4.2 respectively.

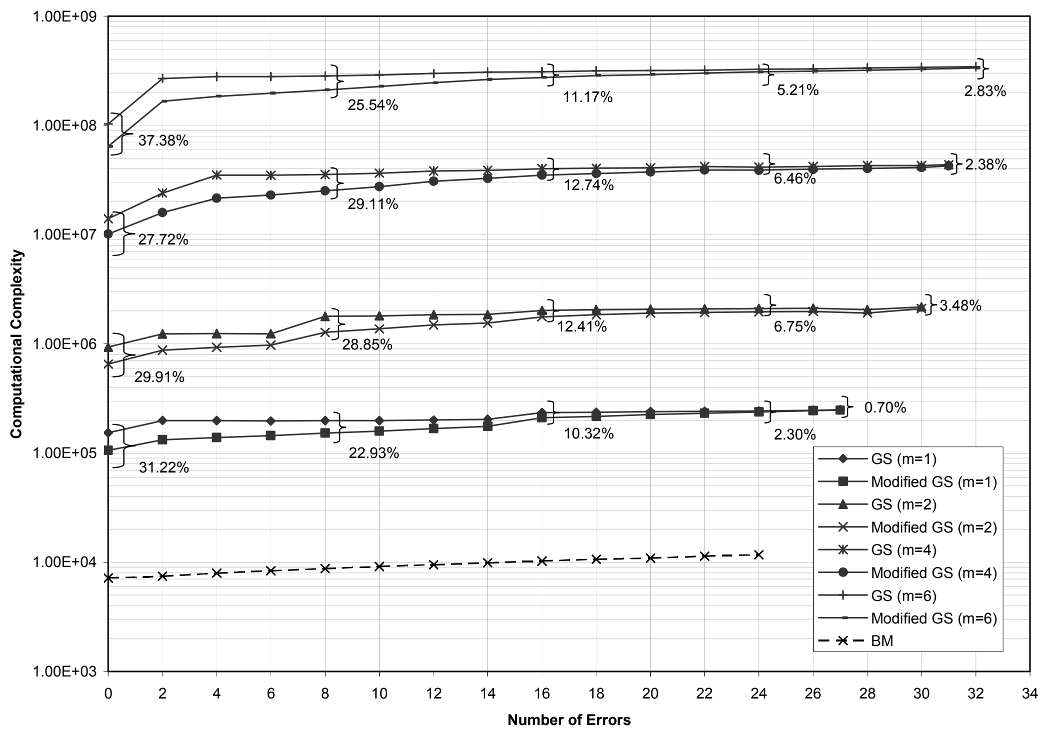
m	$C + 1$	Finite field arithmetic operations
1	64	1.75×10^5
2	190	4.57×10^6
4	631	1.67×10^8
6	1324	1.55×10^9

Table 4.3a Computational complexity for Reed-Solomon code (63, 15)

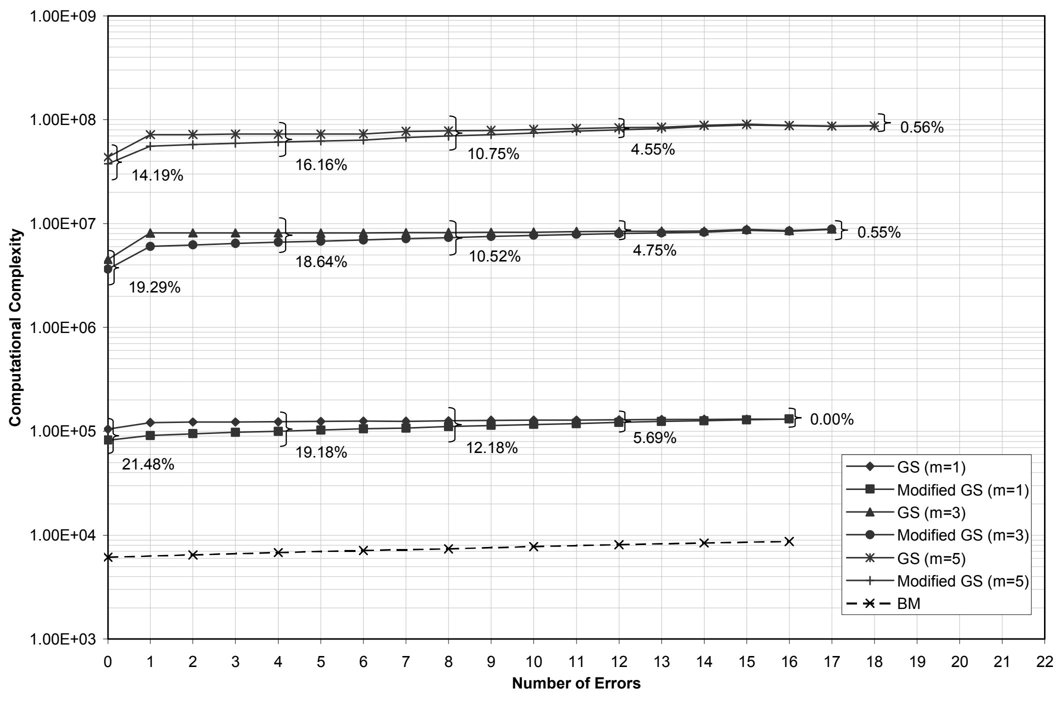
m	$C+1$	Finite field arithmetic operations
1	64	1.75×10^{5}
3	379	3.63×10^{7}
5	946	5.64×10^{8}

Table 4.3b Computational complexity for Reed-Solomon code (63, 31)

In computer simulations, the computational complexity for the two codes has been measured, which are shown in Fig 4.2. Comparing the measurements with Table 4.3, we can see that in most cases (4.47) is a computational complexity upper bound for the GS algorithm and the decoding complexity grows with the error weight. With higher multiplicity *m*, the GS algorithm has better error-correction capability, but at the expense of much higher computation. Comparing the computational complexity between the original and modified GS algorithm, it shows that the lower the error weight, the more computation can be reduced. For Reed-Solomon code (63, 15), the modification can reduce the computational complexity by 37.38% in low error weight situations, but in high error weight situations the complexity is only reduced by 0.70%. For Reed-Solomon code (63, 31), the complexity reduction varies from 21.48% to 0.00% with increasing error weight. In Fig 4.2, the conventional decoding algorithm – Berlekamp-Massey (BM) algorithm's decoding complexity is also measured against the error weight. It can be seen that the conventional decoder's efficiency is still higher than the modified GS algorithm. Therefore, the modified GS algorithm can outperform the BM algorithm in terms of error correction capability, but its decoding complexity is higher. While comparing the modified GS algorithm with the GS algorithm, its decoding efficiency is higher, but error correction capability remains the same.



(a) Reed-Solomon code (63, 15)



(b) Reed-Solomon code (63, 31)

Figure 4.2 Computational complexity analyses for the modified GS algorithm

The modification's error dependent property is analysed as followed.

During the iterative interpolation process, if we define the maximal leading order of the polynomial group G_{i_k} as:

$$\text{maxlod}(G_{i_k}) = \max\{\text{lod}(Q_{i_k,j}) \mid Q_{i_k,j} \in G_{i_k}\} \tag{4.48}$$

The modification (4.39) will start to act when maxlod(G_{i_k}) > C. We use i_a to denote the iterative index when the modification starts to act, which can be explained as:

$$i_a = \{i_k \mid \text{maxlod}(G_{i_k}) > C \text{ and } \text{maxlod}(G_{i_k-1}) \leq C\} \tag{4.49}$$

i_a is error dependent. Under two different situations with error weight e_1 and e_2 (e_1, e_2 $\leq \tau_m$), decoding the same code with multiplicity m, we have:

$$i_a(e_1) \leq i_a(e_2), \text{ if } e_1 \leq e_2 \tag{4.50}$$

which means the lower the error weight, the earlier the modification starts to act.

It has been observed that $Q_{i_k,0}$ is always the first polynomial in the polynomial group to have leading order over C. Therefore, analysing the leading order increase pattern of polynomial $Q_{i_k,0}$ is useful to explain the modified algorithm's error dependent property (4.50). According to the polynomials' property (4.28) and the leading order increase relationship (4.35), we can see that, during the iterative process, $Q_{i_k,0}$'s leading monomial M_L always satisfies $\deg_y(M_L) = 0$ and (4.35) can be simplified for $Q_{i_k,0}$ as:

$$\text{lod}(Q_{i_k+1,0}) = \text{lod}(Q_{i_k,0}) + \left\lfloor \frac{\deg_x Q_{i_k,0}}{k-1} \right\rfloor + 1 \tag{4.51}$$

At the beginning of the iterative process, lod($Q_{0,0}$) = 0. From (4.51), we can see that $Q_{i_k,0}$ will be modified by (4.32) with lod($Q_{i_k+1,0}$) = lod($Q_{i_k,0}$) + 1 for k - 1 times until $\deg_x(Q_{i_k,0}) = k - 1$. Followed by that, $Q_{i_k,0}$ will again be modified by (4.32) with lod($Q_{i_k+1,0}$) = lod($Q_{i_k,0}$) + 2 for k - 1 times until $\deg_x(Q_{i_k,0}) = 2(k - 1)$. This periodic process continues and the leading order of $Q_{i_k,0}$ is accumulated as $1(k - 1) + 2(k - 1) + \cdots$. $Q_{i_k,0}$ will be eliminated once its leading order is over C, therefore the periodic process will stop when lod($Q_{i_k+1,0}$) = lod($Q_{i_k,0}$) + λ, where λ is defined as:

$$\lambda = \min\{x \mid (k-1)\sum_{i=1}^{x} i > C\} \tag{4.52}$$

As there are $\frac{(k-1)(m+1)}{2}$ iterative steps for each of the periodic processes, under the zero error situation the upper bound for $i_a(0)$ can be defined as:

$$i_a(0) \le \frac{(k-1)(m+1)}{2}\lambda \tag{4.53}$$

Once $i_a(0)$ has been determined, $i_a(e)$ would always satisfy:

$$i_a(e) \le i_a(0) + e\binom{m+1}{2} \tag{4.54}$$

which means the lower the error weight, the earlier the modification starts to act and more computation can be reduced as a consequence. Table 4.4 shows some experimental data of $i_a(e)$ from the authors' implementation [52] of Reed-Solomon codes (63, 15) and (63, 31), both of which reveal that (4.54) is being observed.

m	C	λ	$i_a(0)$ upper bound	$i_a(0)$	$i_a(1)$	$i_a(2)$	$i_a(3)$	$i_a(4)$	$i_a(5)$
1	63	3	42	36	37	38	39	40	41
2	189	5	105	99	102	105	108	111	114
4	630	10	350	318	328	338	348	358	368
6	1323	14	686	651	672	693	714	735	756

Table 4.4a $i_a(e)$ for Reed-Solomon code (63, 15)

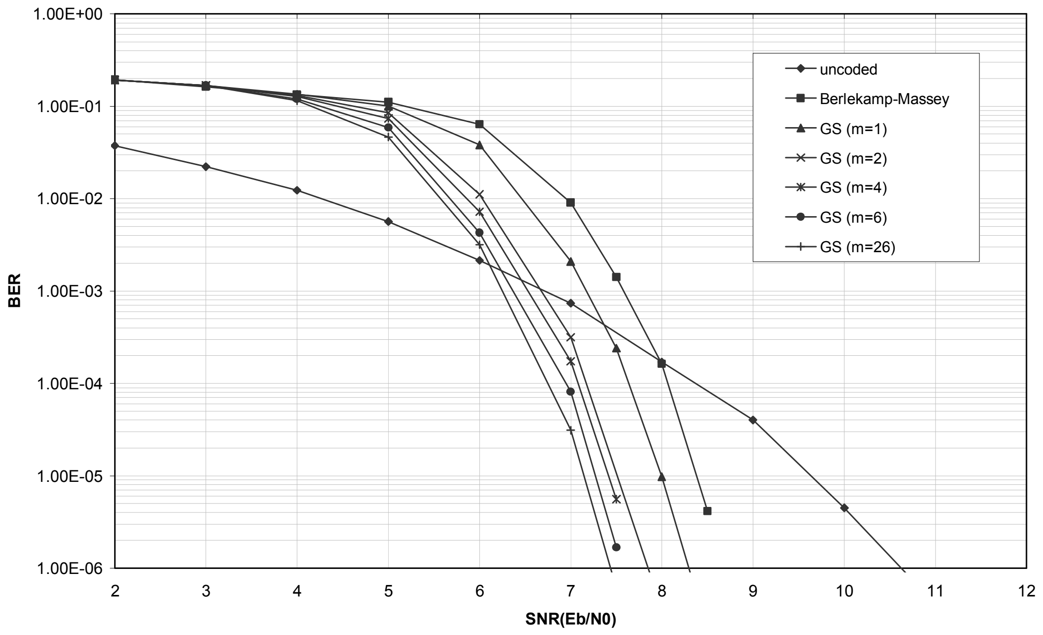
m	C	λ	$i_a(0)$ upper bound	$i_a(0)$	$i_a(1)$	$i_a(2)$	$i_a(3)$	$i_a(4)$	$i_a(5)$
1	63	2	60	47	48	49	50	51	52
3	378	5	300	271	277	283	289	295	301
5	945	8	720	673	688	703	718	733	748

Table 4.4b $i_a(e)$ for Reed-Solomon code (63, 31)

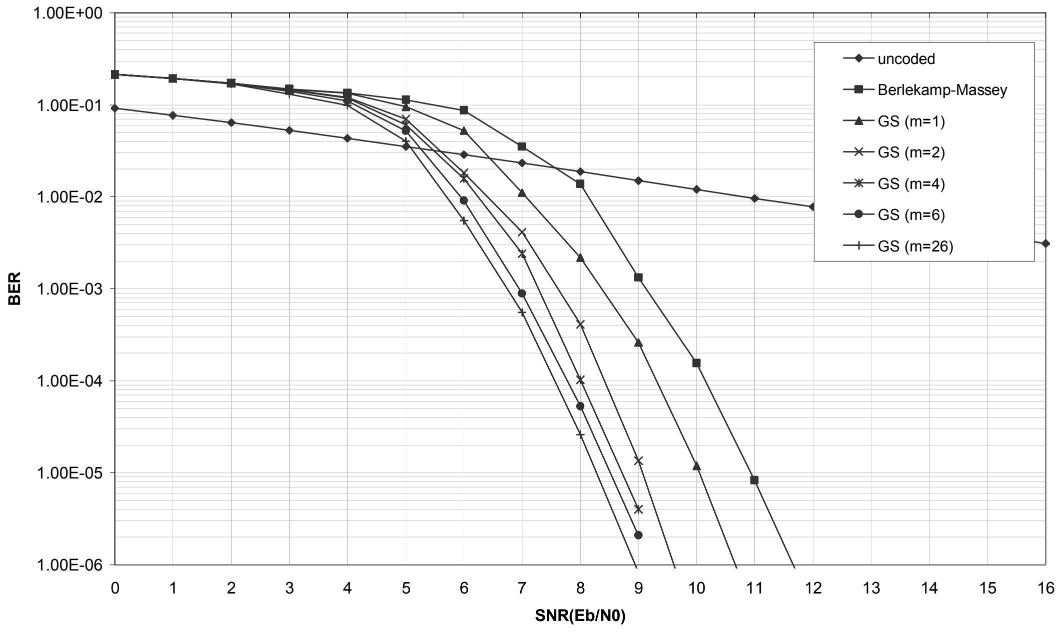
4.6 Simulation Results

A software platform using the C programming language has been developed for the GS algorithm with the complexity reduced modification and a few simulation results have been achieved. The performances of the two Reed-Solomon codes which are defined by examples 4.1 and 4.2 are shown in Fig 4.3 and 4.4. They are also published in [52]. In the simulations, QPSK modulation scheme is employed. The performance of a conventional unique Reed-Solomon code decoding algorithm (Berlekamp-Massey algorithm) is used to compare with the GS algorithm. The Rayleigh fading channel is memoryless with Doppler shift. It is a fast fading channel in which each QPSK symbol is multiplied by a Rayleigh distributed random number with mean value 1.26 and variance 0.5.

Fig 4.3a and 4.3b show the performance of Reed-Solomon code (63, 15) over AWGN and Rayleigh fading channels. Over AWGN channels about 0.4 - 1.3 dB coding gain can be achieved at BER = 10^{-5} with different multiplicity m, while over the Rayleigh fading channels the coding gain is about 1 - 2.8 dB. Fig 4.4a and 4.4b show the performance of Reed-Solomon code (63, 31). For this code, the GS algorithm has no performance advantage with multiplicity m = 1. However with multiplicity m > 1, at BER = 10^{-5} it can achieve 0.2 - 0.8 dB coding gain over AWGN channels and 0.5 - 1.4 dB coding gain over Rayleigh fading channels.

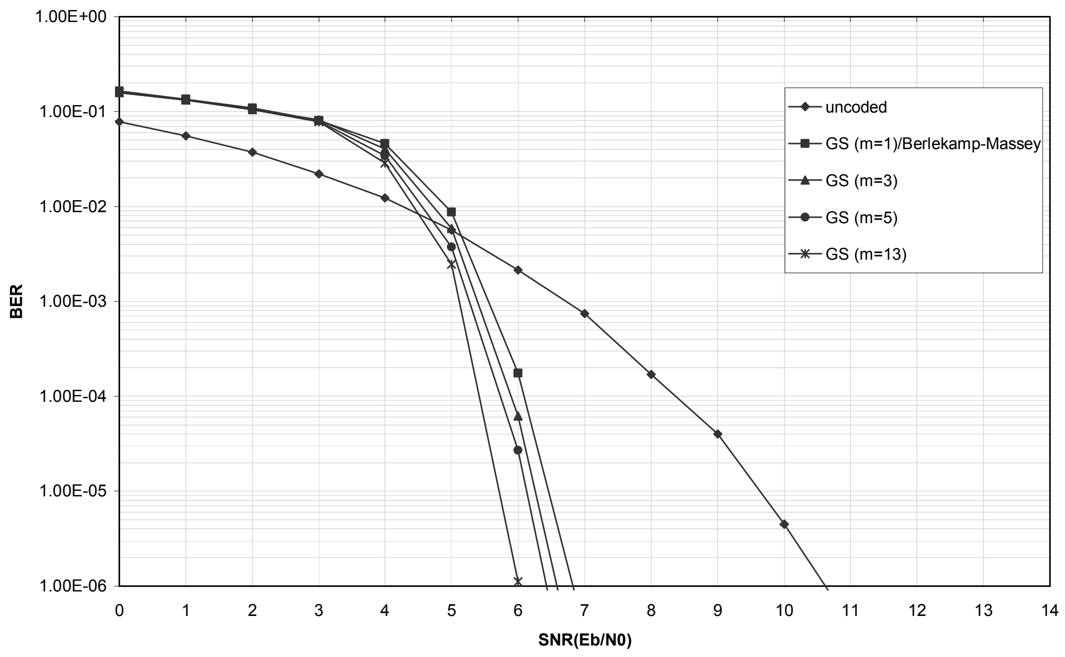


(a) over AWGN channel

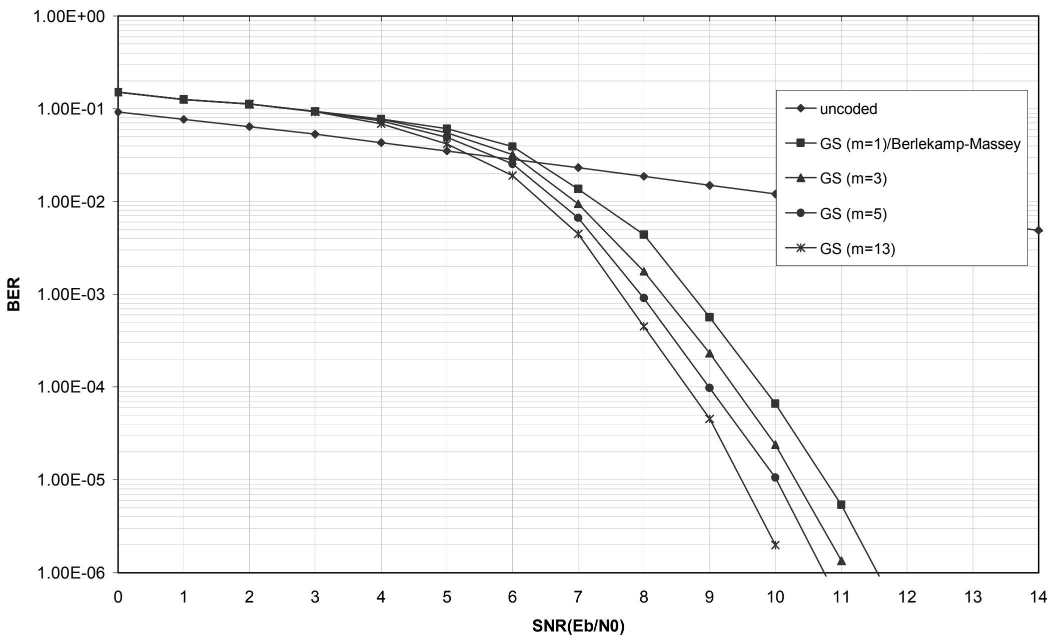


(b) over Rayleigh fading channel

Figure 4.3 Hard-decision list decoding performance of Reed-Solomon code (63, 15)



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 4.4 Hard-decision list decoding performance of Reed-Solomon code (63, 31)

4.7 Conclusion

This chapter explained in detail the hard-decision list decoding algorithm for Reed-Solomon codes. In order to improve the algorithm's decoding efficiency, a novel modification to the interpolation part has been presented. This modification is based on eliminating unnecessary polynomials during the iterative interpolation process. According to the complexity analysis, it can be seen that the decoding complexity is error dependent and the modification can reduce the decoding complexity, especially for low error weight situations in which complexity reduction can be up to 37.38%. Based on this modified GS algorithm, simulation results are presented showing the coding gains over a unique decoding algorithm with more significant gains for low rate codes and in a fading environment. It is very important to point out that this performance advantage is still at the cost of high decoding complexity compared with the unique decoding algorithms. Based on the hard-decision list decoding algorithm, further improvement can be achieved by applying a soft-decision list decoding scheme. This soft-decision list decoding scheme for Reed-Solomon codes is to be presented in the next chapter.

Chapter 5 Soft-Decision List Decoding of Reed-Solomon Codes

5.1 Introduction

From Chapter 4, it was seen that Guruswami and Sudan's hard-decision list decoding scheme can outperform the conventional unique decoding algorithm by correcting errors beyond the half distance boundary. Greater performance improvements can be achieved by employing a soft-decision list decoding algorithm. This chapter presents this soft-decision list decoding algorithm for Reed-Solomon codes. Koetter and Vardy [8] first introduced a soft-decision list decoding scheme for Reed-Solomon codes, which is called the Koetter-Vardy (KV) algorithm. Different to the hard-decision scheme, the soft-decision scheme obtains the received word's posteriori transition probability, which is represented by a reliability matrix $\boldsymbol{\Pi}$. The reliability matrix $\boldsymbol{\Pi}$ is then converted into a multiplicity matrix $\boldsymbol{M}$ based on which the interpolated polynomial is built. The interpolation and factorisation processes are the same as described in Chapter 4. This soft decision scheme can be illustrated by Fig 5.1.

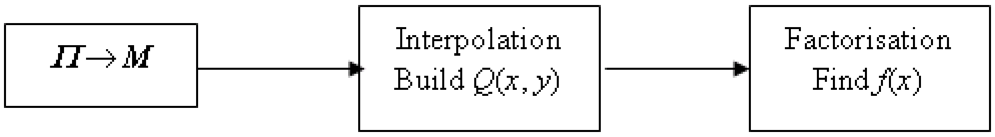


Figure 5.1 Soft-decision list decoding scheme

From Fig 5.1, it can be seen that this soft-decision scheme builds upon the hard-decision scheme with an additional process that converts reliability values into multiplicity values. In [8] it was shown that this soft-decision scheme can outperform Guruswami-Sudan's hard-decision scheme with significant coding gains. [8] also showed that this soft-decision scheme can also outperform the generalised-minimum distance (GMD) decoding algorithm [55]. Some later developments of this soft-decision scheme's application, very large scale integration (VLSI) design and complexity reduction transform are presented in [60, 61] and [50] respectively. This chapter presents the soft-decision list decoding scheme for Reed-Solomon codes. It is shown how to obtain the reliability matrix $\boldsymbol{\Pi}$ based on the received information and how to convert the reliability matrix $\boldsymbol{\Pi}$ into the multiplicity matrix $\boldsymbol{M}$. For the algorithm that converts $\boldsymbol{\Pi}$ to $\boldsymbol{M}$, a practical method to realise the stopping rule based on the designed length of output list is introduced. It is shown in Chapter 4 that the interpolation complexity can be reduced by eliminating polynomials with leading order over the iteration number [52]. This modification scheme can also be applied to the soft-decision scheme. Based on the resulting multiplicity matrix ***M***, the cost of the matrix C_M can be determined, which represents the total iteration number. Therefore, in the following interpolation process, polynomials with leading order greater than this number can be eliminated. It will be shown later in this chapter how the modification performs with the soft-decision scheme. Simulation results of this soft-decision scheme based on assigning the same length of output list as the hard-decision shows the coding gains that can be achieved compared with hard-decision list decoding. More importantly, it is first shown by how much the decoding complexity increases in order to achieve this performance advantage.

5.2 Prerequisite Knowledge

This section gives some prerequisite knowledge for demonstrating the soft-decision list decoding scheme.

Based on the (1, k-1)-weighted degree definition of monomial $x^a y^b$ given in section 4.2.2, let us define the following two parameters:

$$N_{1,k-1}(\delta) = |\{x^a y^b : a, b \geq 0 \text{ and } \deg_{1,k-1}(x^a y^b) \leq \delta, \delta \in \mathrm{N}\}| \tag{5.1}$$

which represents the number of bivariate monomial $x^a y^b$ with (1, k-1)-weighted degree not greater than a nonnegative integer δ [8]. And

$$\Delta_{1,k-1}(v) = \min\{\delta : N_{1,k-1}(\delta) > v, v \in \mathrm{N}\} \tag{5.2}$$

which denotes the minimal value of δ that guarantees $N_{1,k-1}(\delta)$ is greater than a nonnegative integer v [8]. Associated with these two definitions, the following two corollaries shall be proposed.

Corollary 5.1: $\Delta_{1,k-1}(v) = \deg_{1,k-1}(x^a y^b \mid \mathrm{ord}(x^a y^b) = v)$.

Proof: According to section 4.2.2, monomial $x^a y^b$'s (1, k-1)-lexicographic order grows based on its (1, k-1)-weighted degree grows. Up to monomial $x^a y^b$ with ord($x^a y^b$) = v, there are in total v + 1 monomials. Therefore, its (1, k-1)-weighted degree $\deg_{1,k-1}(x^a y^b)$ is the minimal value that guarantees there are more than v monomials with (1, k-1)-weighted degree not greater than it.

Corollary 5.2: $N_{1,k-1}(\delta) > \frac{\delta^2}{2(k-1)}$, and when $\delta \to \infty$, $N_{1,k-1}(\delta) = \frac{\delta^2}{2(k-1)}$ [8].

Proof: To prove corollary 5.2, a (1, k-1)-weighted degree monomial table needs to be taken for analysis. Take Table 4.1a which shows the (1, 4)-weighted degree of monomial $x^a y^b$ as an example. This table can be geometrically plotted as Fig 5.2.

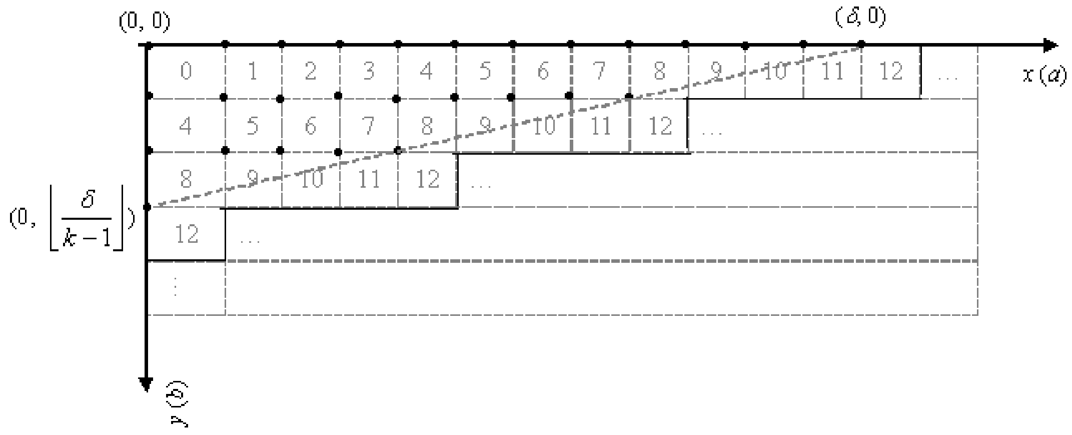


Figure 5.2 Geometric analysis of table 4.1a

In Fig 5.2, index of x-axis and y-axis represent monomial $x^a y^b$'s x degree a and y degree b respectively. The unit distance of x-axis weights 1 and the unit distance of y-axis weights k – 1. Each monomial $x^a y^b$ occupies a unit square and therefore in this figure, $N_{1,k-1}(\delta)$ denotes the total area occupied by monomial $x^a y^b$ with $\deg_{1,k-1}(x^a y^b) \leq \delta$. It is denoted as Area1 = $N_{1,k-1}(\delta)$, which is enclosed by the solid line shown in the figure. The triangle defined by vertexes (0, 0), (δ, 0), and (0, $\left\lfloor \frac{\delta}{k-1} \right\rfloor$) has area $\frac{1}{2}\delta \left\lfloor \frac{\delta}{k-1} \right\rfloor \cong \frac{\delta^2}{2(k-1)}$, which is denoted as Area2 = $\frac{\delta^2}{2(k-1)}$. From Fig 5.2, it is easy to be seen that Area1 > Area 2, and therefore $N_{1,\,k\text{-}1}(\delta) > \frac{\delta^2}{2(k-1)}$. Also, based on the figure, it is not difficult to realise that when $\delta \to \infty$, Area1 and Area2 approach to be equal with each other and therefore $N_{1,\,k\text{-}1}(\delta) = \frac{\delta^2}{2(k-1)}$. Take the case shown in Fig 5.2 as an example, $\delta = 12$, and $N_{1,\,k\text{-}1}(\delta) = N_{1,\,k\text{-}1}(12) = 28$, while $\frac{\delta^2}{2(k-1)} = \frac{12^2}{2\cdot 4} = 18$. Therefore, $N_{1,\,k\text{-}1}(\delta) > \frac{\delta^2}{2(k-1)}$.

5.3 Reliability Information

As mentioned in Chapter 4, the hard-decision decoder obtains a vector of received word $R = (r_0, r_1, \ldots, r_{n\text{-}1})$ ($r_i \in \mathrm{GF}(q)$, $i = 0, 1, \ldots, n - 1$). However, the decoder can also be supplied with posteriori transition probability information [8, 62, 63], or so called the reliability information. The decoder that utilises this information is called soft-decision decoder. This section shows how to obtain the reliability information.

Assume the channel is memoryless with input alphabet $\chi \in \mathrm{GF}(q)$ and output alphabet $\Re \in \mathrm{GF}(q)$. Let Pr indicates the probability function, χ is uniformly distributed over $\mathrm{GF}(q) = (\rho_0, \rho_1, \ldots, \rho_{q\text{-}1})$ as $\Pr(\chi = \rho_0) = \Pr(\chi = \rho_1) = \cdots = \Pr(\chi = \rho_{q\text{-}1})$. If the channel is continuous, then $\Re$ is continuous and $p(\cdot \mid \rho)$ is the probability density function. If the channel is discrete, then $\Re$ is discrete and $p(\cdot \mid \rho)$ is the probability mass function [64]. For the random received alphabet $\Re = \iota_j$ ($j = 0, 1, \ldots, n - 1$), the probability that $\chi = \rho_i$ ($i = 0, 1, \ldots, q - 1$) was being transmitted can be obtained by [8]:

$$\Pr(\chi = \rho_i \mid \Re = \iota_j) = \frac{p(\iota_j \mid \rho_i)\Pr(\chi = \rho_i)}{\sum\limits_{\rho \in GF(q)} p(\iota_j \mid \rho)\Pr(\chi = \rho)} = \frac{p(\iota_j \mid \rho_i)}{\sum\limits_{\rho \in GF(q)} p(\iota_j \mid \rho)} \tag{5.3}$$

For Reed-Solomon codes, $q = n + 1$. For each random received variable ι_j, q transition probabilities can be obtained as: $\Pr(\chi = \rho_0 \mid \Re = \iota_j)$, $\Pr(\chi = \rho_1 \mid \Re = \iota_j)$, …, and $\Pr(\chi = \rho_{q\text{-}1} \mid \Re = \iota_j)$. Further, for a received vector $\overline{\Re} = (\iota_0, \iota_1, \ldots, \iota_{n\text{-}1})$, a $q \times n$ reliability matrix $\boldsymbol{\Pi}$ can be determined as:

$$\boldsymbol{\Pi} = \begin{bmatrix} \pi_{0,0} & \pi_{0,1} & \cdots & \cdots & \cdots & \pi_{0,n-1} \\ \pi_{1,0} & \pi_{1,1} & & & & \pi_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & \pi_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ \pi_{q-1,0} & \pi_{q-1,1} & \cdots & \cdots & \cdots & \pi_{q-1,n-1} \end{bmatrix} \tag{5.4}$$

where its entry $\pi_{i,j}$ is:

$$\pi_{i,j} = \Pr(\chi = \rho_i \mid \Re = \iota_j) \ (i = 0, 1, \ldots, q - 1 \text{ and } j = 0, 1, \ldots, n - 1) \tag{5.5}$$

Referring to Fig 5.1, matrix $\boldsymbol{\Pi}$ is taken as an input to the soft-decision decoder and converted to a multiplicity matrix $\boldsymbol{M}$, followed by the interpolation and factorisation processes.

In the following description, the channel is assumed to be continuous. In order to obtain the reliability information, it is worthy to mention the probability density function. Let us assume S_u are the modulated symbols (e. g. in BPSK, they are S_1, S_2; in QPSK, they are S_1, S_2, S_3, S_4) and any symbol S_u can be projected into V basis functions as S_{u1}, S_{u2}, …, S_{uV}. Given received symbol y, the probability density function with respect to the basis functions (y_t, S_{ut}) can be given as [64]:

$$p(y_t \mid S_{ut}) = \frac{1}{\sqrt{\pi N_0}} \exp\left[-\frac{(y_t - S_{ut})^2}{N_0} \right] \tag{5.6}$$

where N_0 denotes the power of noise. Then the probability density function with respect to symbols (y, S_u) can be calculated by:

$$p(y \mid S_u) = \prod_{t=1}^{V} p(y_t \mid S_{ut}) = \frac{1}{(\pi N_0)^{V/2}} \exp\left[\sum_{t=1}^{V} \frac{-(y_t - S_{ut})^2}{N_0} \right] \tag{5.7}$$

Based on (5.7), the reliability information with respect to finite field alphabets can be further obtained by applying (5.3). Here shows an example of obtaining reliability information based on the received symbols.

Example 5.1 Calculate all the reliability information of received alphabet symbol ι_j which is defined in GF(16). The QPSK modulation is used with mapping scheme shown by Fig 5.3. As each QPSK symbol carries two binary bits information and a GF(16) symbol contains four binary bits , ι_j can be obtained by demodulating two QPSK symbols. The two received QPSK symbols are given as: $y_A = (y_{AI}, y_{AQ})$ = (0.510761, 1.925977) and $y_B = (y_{BI}, y_{BQ})$ = (1.733793, -0.745044). Given Signal-to-Noise Ratio (SNR) = 3 dB and bit energy (E_b) = 0.5, the noise power can be calculated by $N_0 = 0.5/10^{0.3} = 0.250594$.

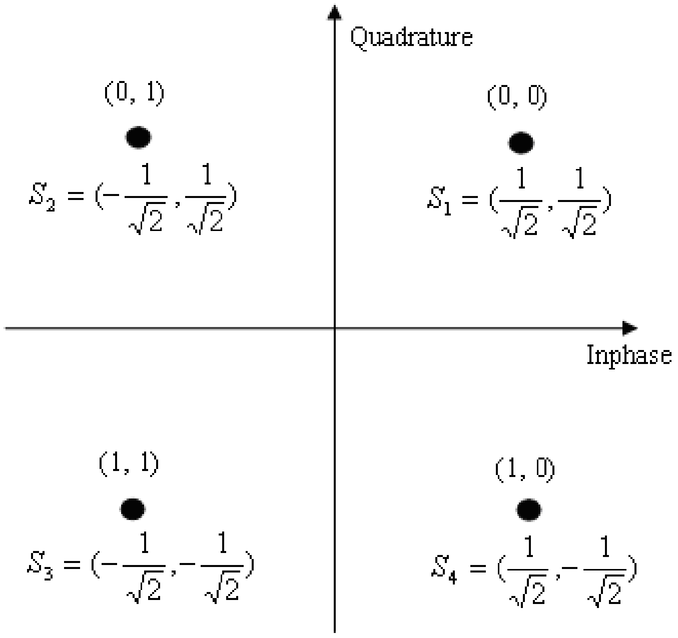

Quadrature
Inphase
(0, 1)
$S_2 = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$
(0, 0)
$S_1 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$
(1, 1)
$S_3 = (-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$
(1, 0)
$S_4 = (\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$

Figure 5.3 QPSK modulation mapping scheme

Applying hard-decision to the received symbols y_A and y_B, they can be demodulated as 00 and 10 respectively by applying the mapping scheme shown in Fig 5.3. Combining them as 0010 and it is equivalent to finite field alphabet 2 and $\iota_j = 2$.

For soft-decision, as it is QPSK modulation, applying (5.7) with $V = 2$ (two basis functions, Inphase and Quadrature). For received symbol y_A, four probability density functions can be obtained as:

$$p(y_A \mid S_1) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{AI} - S_{1I})^2 + (y_{AQ} - S_{1Q})^2}{N_0}\right] = 0.002899,$$

$$p(y_A \mid S_2) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{AI}-S_{2I})^2+(y_{AQ}-S_{2Q})^2}{N_0}\right] = 0.000009,$$

$$p(y_A \mid S_3) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{AI}-S_{3I})^2+(y_{AQ}-S_{3Q})^2}{N_0}\right] = 3.307612 \times 10^{-15},$$

$$p(y_A \mid S_4) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{AI}-S_{4I})^2+(y_{AQ}-S_{4Q})^2}{N_0}\right] = 8.632852 \times 10^{-13},$$

and their summation $\sum_{u=1}^{4} p(y_A \mid S_u) = 0.002908$. Therefore,

$$\Pr(S_1 \mid y_A) = \frac{p(y_A \mid S_1)}{\sum_{u=1}^{4} p(y_A \mid S_u)} = 0.996905,\ \Pr(S_2 \mid y_A) = \frac{p(y_A \mid S_2)}{\sum_{u=1}^{4} p(y_A \mid S_u)} = 0.003095,$$

$$\Pr(S_3 \mid y_A) = \frac{p(y_A \mid S_3)}{\sum_{u=1}^{4} p(y_A \mid S_u)} = 1.137418 \times 10^{-12},$$

$$\Pr(S_4 \mid y_A) = \frac{p(y_A \mid S_4)}{\sum_{u=1}^{4} p(y_A \mid S_u)} = 2.968656 \times 10^{-10}.$$

For received symbol y_B, four probability density functions can be obtained as:

$$p(y_B \mid S_1) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{BI}-S_{1I})^2+(y_{BQ}-S_{1Q})^2}{N_0}\right] = 4.194589 \times 10^{-6},$$

$$p(y_B \mid S_2) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{BI}-S_{2I})^2+(y_{BQ}-S_{2Q})^2}{N_0}\right] = 1.050284 \times 10^{-14},$$

$$p(y_B \mid S_3) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{BI}-S_{3I})^2+(y_{BQ}-S_{3Q})^2}{N_0}\right] = 5.980199 \times 10^{-11},$$

$$p(y_B \mid S_4) = \frac{1}{\pi N_0}\exp\left[-\frac{(y_{BI}-S_{4I})^2+(y_{BQ}-S_{4Q})^2}{N_0}\right] = 0.018812,$$

and their summation $\sum_{u=1}^{4} p(y_B \mid S_u) = 0.018816$. Therefore,

$\Pr(S_1 \mid y_B) = \dfrac{p(y_B \mid S_1)}{\sum_{u=1}^{4} p(y_B \mid S_u)} = 2.229267 \times 10^{-4}$,

$\Pr(S_2 \mid y_B) = \dfrac{p(y_B \mid S_2)}{\sum_{u=1}^{4} p(y_B \mid S_u)} = 5.581866 \times 10^{-13}$,

$\Pr(S_3 \mid y_B) = \dfrac{p(y_B \mid S_3)}{\sum_{u=1}^{4} p(y_B \mid S_u)} = 3.178252 \times 10^{-9}$, $\Pr(S_4 \mid y_B) = \dfrac{p(y_B \mid S_4)}{\sum_{u=1}^{4} p(y_B \mid S_u)} = 0.999787$.

Based on the above reliability values obtained from two received QPSK symbols, the reliability values for received alphabet symbol ι_j can be calculated as:

$\Pr(\chi = 0 \mid \Re = \iota_j) = \Pr(0000 \mid \Re = \iota_j) = \Pr(S_1 \mid y_A) \cdot \Pr(S_1 \mid y_B) = 2.222367 \times 10^{-14}$,

$\Pr(\chi = 1 \mid \Re = \iota_j) = \Pr(0001 \mid \Re = \iota_j) = \Pr(S_1 \mid y_A) \cdot \Pr(S_2 \mid y_B) = 5.564590 \times 10^{-13}$,

$\Pr(\chi = 2 \mid \Re = \iota_j) = \Pr(0010 \mid \Re = \iota_j) = \Pr(S_1 \mid y_A) \cdot \Pr(S_4 \mid y_B) = 0.996693$,

$\Pr(\chi = 3 \mid \Re = \iota_j) = \Pr(0011 \mid \Re = \iota_j) = \Pr(S_1 \mid y_A) \cdot \Pr(S_3 \mid y_B) = 3.168415 \times 10^{-9}$,

$\Pr(\chi = 4 \mid \Re = \iota_j) = \Pr(0100 \mid \Re = \iota_j) = \Pr(S_2 \mid y_A) \cdot \Pr(S_1 \mid y_B) = 6.899581 \times 10^{-7}$,

$\Pr(\chi = 5 \mid \Re = \iota_j) = \Pr(0101 \mid \Re = \iota_j) = \Pr(S_2 \mid y_A) \cdot \Pr(S_2 \mid y_B) = 1.727588 \times 10^{-15}$,

$\Pr(\chi = 6 \mid \Re = \iota_j) = \Pr(0110 \mid \Re = \iota_j) = \Pr(S_2 \mid y_A) \cdot \Pr(S_4 \mid y_B) = 0.003094$,

$\Pr(\chi = 7 \mid \Re = \iota_j) = \Pr(0111 \mid \Re = \iota_j) = \Pr(S_2 \mid y_A) \cdot \Pr(S_3 \mid y_B) = 9.741342 \times 10^{-12}$,

$\Pr(\chi = 8 \mid \Re = \iota_j) = \Pr(1000 \mid \Re = \iota_j) = \Pr(S_4 \mid y_A) \cdot \Pr(S_1 \mid y_B) = 6.617927 \times 10^{-14}$,

$\Pr(\chi = 9 \mid \Re = \iota_j) = \Pr(1001 \mid \Re = \iota_j) = \Pr(S_4 \mid y_A) \cdot \Pr(S_2 \mid y_B) = 1.657064 \times 10^{-22}$,

$\Pr(\chi = 10 \mid \Re = \iota_j) = \Pr(1010 \mid \Re = \iota_j) = \Pr(S_4 \mid y_A) \cdot \Pr(S_4 \mid y_B) = 2.968024 \times 10^{-10}$,

$\Pr(\chi = 11 \mid \Re = \iota_j) = \Pr(1011 \mid \Re = \iota_j) = \Pr(S_4 \mid y_A) \cdot \Pr(S_3 \mid y_B) = 9.435137 \times 10^{-19}$,

$\Pr(\chi = 12 \mid \Re = \iota_j) = \Pr(1100 \mid \Re = \iota_j) = \Pr(S_3 \mid y_A) \cdot \Pr(S_1 \mid y_B) = 2.535608 \times 10^{-16}$,

$\Pr(\chi = 13 \mid \Re = \iota_j) = \Pr(1101 \mid \Re = \iota_j) = \Pr(S_3 \mid y_A) \cdot \Pr(S_2 \mid y_B) = 6.348915 \times 10^{-25}$,

$\Pr(\chi = 14 \mid \Re = \iota_j) = \Pr(1110 \mid \Re = \iota_j) = \Pr(S_3 \mid y_A) \cdot \Pr(S_4 \mid y_B) = 1.137176 \times 10^{-12}$,

$\Pr(\chi = 15 \mid \Re = \iota_j) = \Pr(1111 \mid \Re = \iota_j) = \Pr(S_3 \mid y_A) \cdot \Pr(S_3 \mid y_B) = 3.615001 \times 10^{-21}$.

Among the above reliability values, $\Pr(\chi = 2 \mid \Re = \iota_j)$ is the maximal. This indicates that received finite field alphabet ι_j has the highest probability of being transmitted as $\chi = 2$, which matches the hard-decision result mentioned above.

5.4 From Reliability Values to Multiplicity Values

The reliability matrix $\boldsymbol{\Pi}$ (5.4) is then converted into a multiplicity matrix $\boldsymbol{M}$ which defines the interpolated points with corresponding interpolation multiplicities. One of the core contributions of [8] is presenting the algorithm that converts $\boldsymbol{\Pi}$ to $\boldsymbol{M}$. This algorithm is described as followed:

Algorithm 5.1: Convert reliability matrix $\boldsymbol{\Pi}$ to multiplicity matrix $\boldsymbol{M}$.

```
Input: Reliability matrix $\boldsymbol{\Pi}$ and a desired value of $s = \sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}$
Initialisation: Set $\boldsymbol{\Pi}^* = \boldsymbol{\Pi}$ and $q \times n$ all-zero multiplicity matrix $\boldsymbol{M}$
(i): While (s > 0) {
(ii): Find the maximal entry $\pi^*_{i,j}$ in $\boldsymbol{\Pi}^*$ with position $(i, j)$
(iii): Update $\pi^*_{i,j}$ in $\boldsymbol{\Pi}^*$ as $\pi^*_{i,j} = \frac{\pi_{i,j}}{m_{i,j} + 2}$
(iv): Update $m_{i,j}$ in $\boldsymbol{M}$ as $m_{i,j} = m_{i,j} + 1$
(v): $s = s - 1$
}
```

Algorithm 5.1 results a $q \times n$ multiplicity matrix $\boldsymbol{M}$ which can be written as:

$$\boldsymbol{M} = \begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & \cdots & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & & & & m_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & m_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ m_{q-1,0} & m_{q-1,1} & \cdots & \cdots & \cdots & m_{q-1,n-1} \end{bmatrix} \tag{5.8}$$

in which its entries $m_{i,j}$ represents the multiplicity value of interpolated point (x_j, ρ_i) (j = 0, 1, …, n – 1 and i = 0, 1, …, q - 1). x_j are the finite field elements used in encoding (3.14). In algorithm 5.1, desired value s indicates the total value of multiplicity of all interpolated points. This algorithm gives priority to those interpolated points which correspond to a higher reliability values $\pi_{i,j}$ to be assigned with a higher multiplicity values $m_{i,j}$. For illustration of the algorithm, here gives a work example.

Example 5.2 For soft-decision list decoding of the (7, 2) Reed-Solomon code which is defined in GF(8), the following 8 × 7 reliability matrix $\boldsymbol{\Pi}$ is obtained by the receiver:

$$\boldsymbol{\Pi} = \begin{bmatrix} 0.959796 & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.954880 & 0.000006 \\ 0.009543 & 0.736533 & \underline{0.968097} & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

(Note: in the matrix $\boldsymbol{\Pi}(\boldsymbol{\Pi}^*)$, the maximal entry is underlined)

Apply Algorithm 5.1 with a desired value s = 20.

Initialisation: Set $\boldsymbol{\Pi}^* = \boldsymbol{\Pi}$ and $\boldsymbol{M} = 0$.

As s = 20 > 0, find the maximal entry $\pi_{i,j}{}^* = 0.968097$ in $\boldsymbol{\Pi}^*$ with position $(i, j) = (4, 2)$

Update $\pi_{4,2}{}^*$ as $\pi_{4,2}{}^* = \dfrac{\pi_{4,2}}{m_{4,2}+2} = \dfrac{0.968097}{0+2} = 0.484048$

Update $m_{4,2}$ in $\boldsymbol{M}$ as $m_{4,2} = 0 + 1 = 1$

$s = s - 1 = 19$

Now the updated $\boldsymbol{\Pi}^*$ is:

$$\boldsymbol{\Pi}^{*}=\begin{bmatrix} \underline{0.959796} & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.954880 & 0.000006 \\ 0.009543 & 0.736533 & 0.484048 & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

and the updated $\boldsymbol{M}$ is:

$$\boldsymbol{M}=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

In the next iteration, as $s = 19 > 0$, find the maximal entry $\pi_{i,j}^{*} = 0.959696$ in $\boldsymbol{\Pi}^{*}$ with position $(i, j) = (0, 0)$

Update $\pi_{0,0}^{*}$ as $\pi_{0,0}^{*} = \dfrac{\pi_{0,0}}{m_{0,0}+2} = \dfrac{0.959796}{0+2} = 0.479898$

Update $m_{0,0}$ in $\boldsymbol{M}$ as $m_{0,0} = 0 + 1 = 1$

$s = s - 1 = 18$

Now the updated $\boldsymbol{\Pi}^{*}$ is:

$$\Pi^{*}=\begin{bmatrix} 0.479898 & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & \underline{0.954880} & 0.000006 \\ 0.009543 & 0.736533 & 0.484048 & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

and the updated $\boldsymbol{M}$ is:

$$\boldsymbol{M}=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Following the same process until $s = 0$, the updated Π^{*} is:

$$\Pi^{*}=\begin{bmatrix} 0.239949 & 0.214170 & 0.005453 & 0.230535 & 0.001125 & 0.000505 & 0.230576 \\ 0.001749 & 0.005760 & 0.000000 & 0.175013 & 0.224388 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.238720 & 0.000006 \\ 0.009543 & \underline{0.245511} & 0.242024 & 0.003180 & 0.000000 & 0.000000 & 0.139395 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

and the updated $\boldsymbol{M}$ is:

$$\boldsymbol{M} = \begin{bmatrix} 3 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 3 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

As $s = 0$, the iteration stops and the algorithm outputs matrix $\boldsymbol{M}$ = $\begin{bmatrix} 3 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 3 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$.

In the resulting multiplicity matrix $\boldsymbol{M}$, it can be seen that the sum of its entries $\sum_{i=0}^{7}\sum_{j=0}^{6} m_{i,j} = 20$ which is the desired value s set in the beginning. Also, take any two entries in $\boldsymbol{\Pi}$ for comparison, $\pi_{4,\,3} = 0.968097 > \pi_{4,\,2} = 0.836533$. In the resulting multiplicity matrix $\boldsymbol{M}$, $m_{4,\,3} = 3 \geq m_{4,\,2} = 2$. It indicates that the interpolated point which corresponds to a higher reliability value $\pi_{i,\,j}$ will be assigned with a higher multiplicity value $m_{i,\,j}$.

5.5 Soft-Decision Solution

Based on the multiplicity matrix (5.8), interpolated polynomial $Q(x, y) = \sum_{a,b\in\mathrm{N}} Q_{ab}x^a y^b$ is built so that $Q(x, y)$ has a zero of multiplicity at least $m_{i,j}$ ($m_{i,j} \neq 0$) over interpolated point (x_j, ρ_i). It can be seen that the total number of interpolated points covered by $Q(x, y)$ is:

$$|\{\, m_{i,j} \neq 0 \mid m_{i,j} \in \boldsymbol{M},\, i = 0, 1, \ldots, q - 1 \text{ and } j = 0, 1, \ldots, n - 1\}| \qquad (5.9)$$

Based on the derivation of section 4.3.1, with respect to point (x_j, ρ_i), Q's coefficients Q_{ab} should satisfy:

$$\sum_{a\geq\alpha, b\geq\beta} \binom{a}{\alpha}\binom{b}{\beta} Q_{ab} x_j^{a-\alpha} \rho_i^{b-\beta} = 0, \forall\ \alpha, \beta \in \mathrm{N} \text{ and } \alpha + \beta < m_{i,j} \tag{5.10}$$

There are in total:

$$C_M = \frac{1}{2}\sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}(m_{i,j}+1) \tag{5.11}$$

constraints of type (5.10) for Q's coefficients Q_{ab}. C_M is called the cost of multiplicity matrix $\boldsymbol{M}$, which also denotes the number of iteration in the interpolation process.

Referring to encoding process described by equation (3.14), if $x_0, x_1, \ldots, x_{n-1}$ are the finite field elements and $c_0, c_1, \ldots, c_{n-1}$ are the corresponding encoded code word symbols, interpolated polynomial $Q(x, y)$ can be explained as passing through point (x_0, c_0) with multiplicity $m_0 = m_{i,0}$ ($\rho_i = c_0$), point (x_1, c_1) with multiplicity $m_1 = m_{i,1}$ ($\rho_i = c_1$), …, and point (x_{n-1}, c_{n-1}) with multiplicity $m_{n-1} = m_{i,n-1}$ ($\rho_i = c_{n-1}$). Based on lemma 4.1 described in section 4.4.1, if $f(x)$ is the message polynomial (3.13) that satisfies $f(x_j) = c_j$ ($j = 0, 1, \ldots, n - 1$), polynomial $Q(x, f(x))$ should satisfy:

$$(x-x_0)^{m_0}(x-x_1)^{m_1}\cdots(x-x_{n-1})^{m_{n-1}} \mid Q(x, f(x)) \tag{5.12}$$

Again, if we let $g_1(x) = (x-x_0)^{m_0}(x-x_1)^{m_1}\cdots(x-x_{n-1})^{m_{n-1}}$ and $g_2(x) = Q(x, f(x))$, based on (5.12), $g_1(x) \mid g_2(x)$. $g_1(x)$ has x degree $\deg_x(g_1(x)) = m_0 + m_1 + \cdots + m_{n-1}$. The x degree of $g_1(x)$ is defined as the code word score with respect to multiplicity matrix $\boldsymbol{M}$ as:

$$S_M(\bar{c}) = \deg_x(g_1(x)) = m_0 + m_1 + \cdots + m_{n-1} = \sum_{j=0}^{n-1}\{m_{i,j} \mid \rho_i = c_j, i = 0, 1, \ldots, q-1\} \tag{5.13}$$

The x degree of $g_2(x)$ is bounded by $\deg_x(g_2(x)) \leq \deg_{1,k-1}Q(x, y)$. Therefore, if $S_M(\bar{c}) > \deg_{1,k-1}Q(x, y)$, then $\deg_x(g_1(x)) > \deg_x(g_2(x))$. To satisfy both $\deg_x(g_1(x)) > \deg_x(g_2(x))$ and $g_1(x) \mid g_2(x)$, the only solution is: $g_2(x) = 0$ which indicates $Q(x, f(x)) = 0$ or equivalently $y - f(x) \mid Q(x, y)$, and message polynomial $f(x)$ can be found out by determining $Q(x, y)$'s y roots. As a result, drawn from [8], the following lemma is proposed.

Lemma 5.3: If code word score with respect to multiplicity matrix $\boldsymbol{M}$ is greater than the interpolated polynomial $Q(x, y)$'s (1, k-1)-weighted degree as:

$$S_M(\bar{c}) > \deg_{1,\,k\text{-}1}Q(x, y)$$

then $Q(x, f(x)) = 0$ or equivalently $y - f(x) \mid Q(x, y)$. Message polynomial $f(x)$ can be found out by determining $Q(x, y)$'s y roots [8].

If the (1, k-1)-weighted degree of interpolated polynomial Q is δ^*, based on (5.1), Q has at most $N_{1,k-1}(\delta^*)$ nonzero coefficients. The interpolation procedure generates a system of C_M linear equations of type (5.10). The system will be solvable if [8]:

$$N_{1,k-1}(\delta^*) > C_M \tag{5.14}$$

Based on (5.2), in order to guarantee the solution, the (1, k-1)-weighted degree δ^* of the interpolated polynomial Q should be large enough so that:

$$\deg_{1,k-1}(Q(x, y)) = \delta^* = \Delta_{1,k-1}(C_M) \tag{5.15}$$

Therefore, according to lemma 5.3, given the soft-decision code word score (5.13) and the (1, k-1)-weighted degree of the interpolated polynomial Q (5.15), message polynomial f can be found out if:

$$S_M(\bar{c}) > \Delta_{1,k-1}(C_M) \tag{5.16}$$

As the (1, k-1)-weighted degree of the interpolated polynomial $Q(x, y)$ can be determined by (5.15) while $\Delta_{1,k-1}(C_M)$ can be realised by applying corollary 5.1 as: $\Delta_{1,k-1}(C_M) = \deg_{1,\,k\text{-}1}(x^a y^b \mid \text{ord}(x^a y^b) = C_M)$, a stopping rule for algorithm 5.1 based on the designed length of output list l can be imposed. This is more realistic for assessing soft-decision list decoding scheme's performance. As factorisation outputs are the y roots of the interpolated polynomial Q, the maximal number of outputs l_M based on the interpolated polynomial Q is:

$$l_M = \deg_{0,1} Q(x, y) = \left\lfloor \frac{\deg_{1,k-1} Q(x,y)}{k-1} \right\rfloor = \left\lfloor \frac{\Delta_{1,k-1}(C_M)}{k-1} \right\rfloor \tag{5.17}$$

Therefore, after step (v) of algorithm 5.1, the updated multiplicity matrix ***M***'s cost C_M can be determined using (5.11). As C_M has been determined, the interpolated polynomial $Q(x, y)$'s (1, k-1)-weighted degree can be determined by (5.15). Then (5.17) can be applied to calculate the maximal number of factorisation outputs l_M. Based on a designed length of output list l, stop algorithm 5.1 once l_M is greater then l.

From the above description, when designed length of output list $l \to \infty$, the soft-decision's asymptotically optimal result can be achieved as a high enough code word score $S_M(\bar{c})$ can always be produced to satisfy condition (5.16). When $l \to \infty$, $s \to \infty$, the cost of multiplicity matrix $C_M \to \infty$ so as $\Delta_{1,k-1}(C_M) \to \infty$. Based on corollary 5.2, we have $\Delta_{1,k-1}(C_M) = \sqrt{2(k-1)N_{1,k-1}(\Delta_{1,k-1}(C_M))} = \sqrt{2(k-1)C_M}$. Successful list decoding coding condition (5.16) can be written as:

$$S_M(\bar{c}) > \sqrt{2(k-1)C_M} \tag{5.18}$$

or equivalently,

$$\sum_{j=0}^{n-1} \{m_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} > \sqrt{(k-1)\sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}(m_{i,j}+1)} \tag{5.19}$$

As $m_{i,j} \to \infty$, to access equation (5.19), the following lemma is needed.

Lemma 5.4 For algorithm 5.1, when $s \to \infty$, $\frac{\pi_{i,j}}{n} \cong \frac{m_{i,j}}{s}$ [8].

Based on lemma 5.4, we have $m_{i,j} = \frac{s}{n}\pi_{i,j}$ and substitute it into (5.19), (5.19) can be re-written as:

$$\frac{s}{n}\sum_{j=0}^{n-1} \{\pi_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} > \frac{s}{n}\sqrt{(k-1)\sum_{i=0}^{q-1}\sum_{j=0}^{n-1} \pi_{i,j}(\pi_{i,j}+\frac{n}{s})} \tag{5.20}$$

As when $s \to \infty$, $\frac{n}{s} \cong 0$, (5.20) can be further approximated as:

$$\sum_{j=0}^{n-1} \{\pi_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} > \sqrt{(k-1)\sum_{i=0}^{q-1}\sum_{j=0}^{n-1} \pi_{i,j}{}^2} \tag{5.21}$$

Therefore, soft-decision list decoding's optimal result is determined by the received reliability values. In practice, due to the decoding complexity restriction, soft-decision list decoding can only perform based on a designed length of output list *l*. This output length restriction in fact leads to practical decoding performance degradation. This phenomenon can be seen later when the simulation results are discussed. Also, based on equation (5.21), soft-decision list decoding has more performance improvement potential for low rate codes as the *k* value is small.

5.6 Complexity reduction Interpolation and Factorisation

Based on the multiplicity matrix ***M***, the following complexity reduction interpolation and factorisation processes can be implemented by applying algorithm 4.1 and algorithm 4.2 respectively. The interpolated polynomial $Q(x, y)$ builds upon the multiplicity matrix ***M*** with a zero of multiplicity at least $m_{i,j}$ ($m_{i,j} \neq 0$) at all the associated points (x_j, ρ_i) ($j = 0, 1, \ldots, n-1$ and $i = 0, 1, \ldots, q-1$). As details of the interpolation and factorisation processes can be referred to Chapter 4 or the author's publication [52, 53], this section only mentions some necessary modifications to the interpolation process, while factorisation process remains the same. Also, an analysis of how much decoding complexity can be reduced by the modification scheme proposed by section 4.3.3 will be given.

As mentioned in section 5.5, to build interpolated polynomial $Q(x, y)$, there are in total C_M (5.11) iterations. Therefore, the iteration index i_k used in algorithm 4.1 is: $i_k = 0, 1, \ldots, C_M$. Based on a designed length of output list *l*, the initialisation at step (i) can be modified as:

$$G_0 = \{Q_{0,j} = y^j, j = 0, 1, \ldots, l\} \tag{5.22}$$

As there are in total C_M iterations, based on the complexity reduction scheme's description given in section 4.3.3, the interpolated polynomial Q's leading order is less than or equal to the total number of iterations C_M as:

$$\mathrm{lod}(Q(x, y)) \leq C_M \tag{5.23}$$

This indicates the fact that (5.15) is an upper bound for the interpolated polynomial's (1, k-1)-weighted degree as:

$$\deg_{1,\, k\text{-}1} Q(x, y) \leq \Delta_{1,k-1}(C_M) \tag{5.24}$$

Based on (5.23), those polynomials with leading order over C_M will neither be chosen as the interpolated polynomial, nor be modified with the interpolated polynomial. Therefore, they can be eliminated from the polynomial group and the modification at step (ii) can be re-written as:

$$G_{i_k} = \{Q_{i_k,j} \mid \mathrm{lod}(Q_{i_k,j}) \leq C_M\} \tag{5.25}$$

With respect to interpolated point (x_j, ρ_i) and Hasse derivative parameter (α, β) ($\alpha + \beta < m_{i,j}$), the Hasse derivative evaluation performed at step (iii) of algorithm 4.1 can be modified and determined by equation (5.10). For the following of the process, it is the same as algorithm 4.1 only take a notice that for polynomial modification (4.32), the interpolated point's x-coordinate x_i should be replaced by the x_j which is the current interpolated point's x-coordinate. And also, distinguish index j for interpolated point's x-coordinate x_j and polynomials $Q_{i_k,j}$ in the group G_{i_k}.

As discussed in section 4.5, the complexity reduction scheme is error dependent that it can reduce interpolation complexity more significantly in low error weight situations [52]. However, in a soft-decision decoder, no hard-decision received vector is obtained and therefore it is impossible to measure the actual error weight (Hamming distance between the received word and transmitted code word). In order to evaluate the complexity modification scheme's performance for this soft-decision list decoder, decoding complexity of the original and modified interpolation processes is measured against the SNR values. Fig 5.4 shows how much decoding complexity can be reduced for soft-decision list decoding of Reed-Solomon code (63, 15) with output length l = 2 and 4. From Fig 5.4 it can be seen that more decoding complexity can be reduced in high SNR values which in fact correspond to low error weight situations if hard-decision was made. Under high SNR values, complexity can be reduced up to 36.45%. However, in low SNR values, complexity reduction is not as significant.

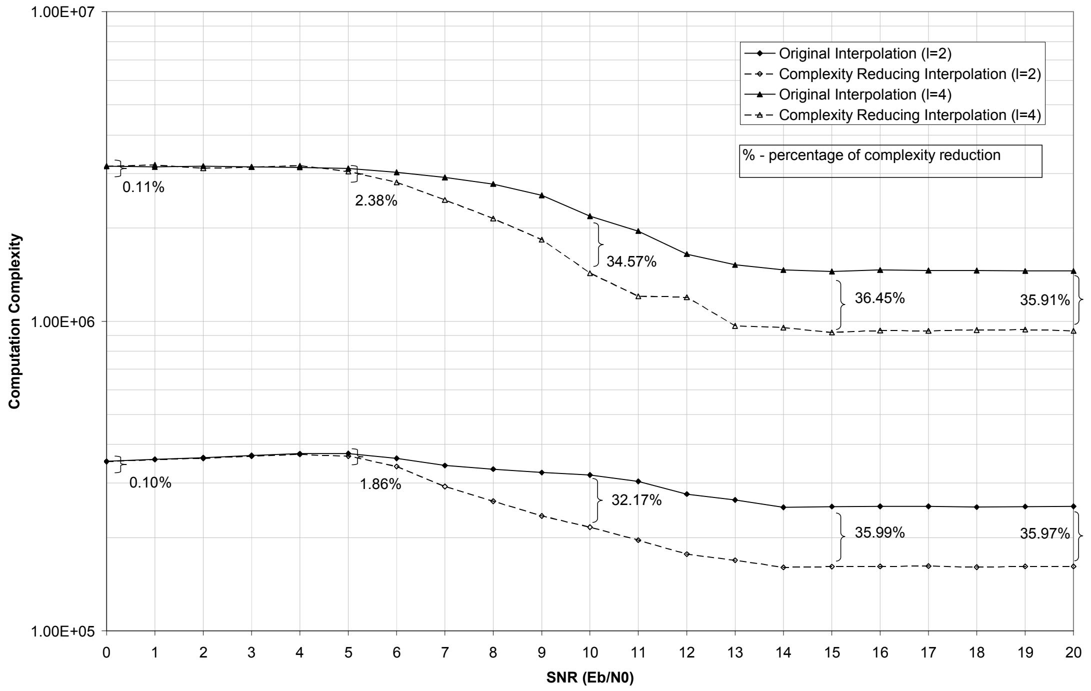
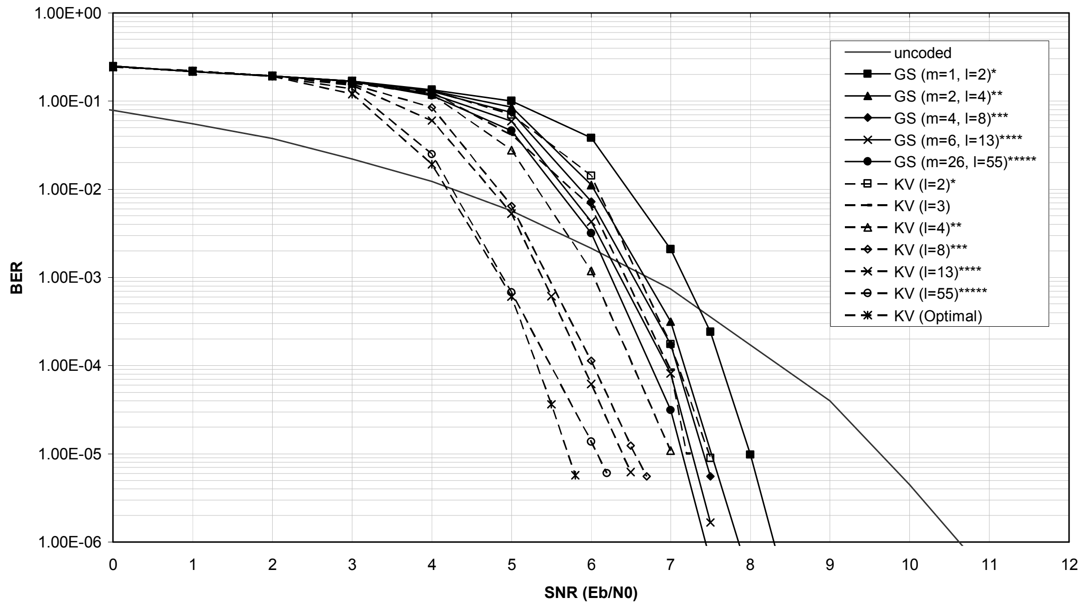


Figure 5.4 Complexity reduction analysis for soft-decision list decoding of Reed-Solomon code (63, 15)

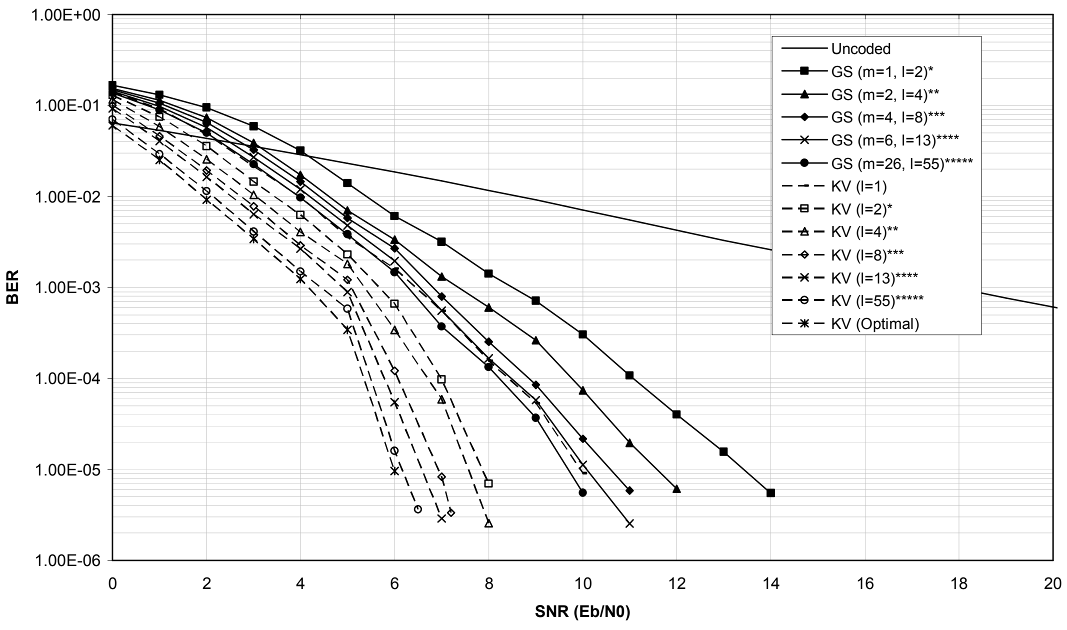
5.7 Simulation Results Discussion

This section presents soft-decision list decoding results for the two Reed-Solomon codes introduced in Chapter 4: Reed-Solomon codes (63, 15) and (63, 31). They are simulated under both AWGN and Rayleigh fading channels. The Rayleigh fading channel is frequency nonselective with Doppler frequency [64] 126.67 Hz and data rate 30 kb/s. The fading profile is generated using Jakes' method [64]. The fading coefficients have mean value 1.55 and variance 0.60. Under Rayleigh fading channel, a block interleaver with size 63 × 63 is used to combat the fading effect. QPSK modulation scheme is used and simulations are run using C programming language.

Soft-decision list decoding results for Reed-Solomon codes (63, 15) and (63, 31) are shown by Fig 5.5 and 5.6 respectively. Soft-decision's performance is compared with the hard-decision based on giving the same maximal length of output list l (or so called the designed length of output list for soft-decision), which is indicated by different patterns of '*' in the figures.

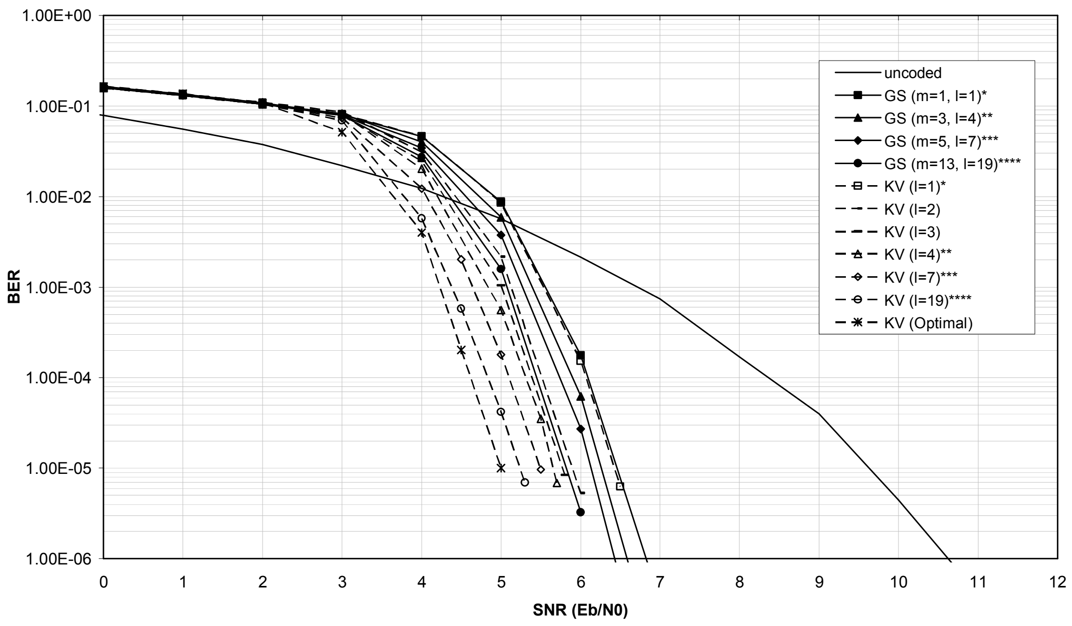


(a) over AWGN channel

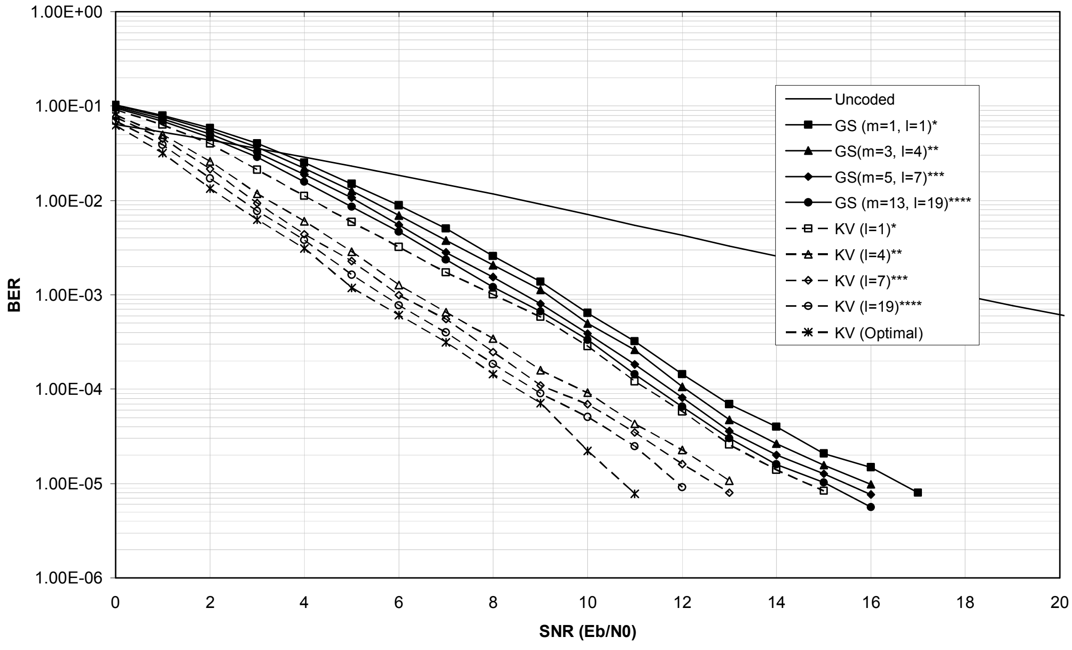


(b) over Rayleigh fading channel

Figure 5.5 Soft-decision list decoding of Reed-Solomon code (63, 15)



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 5.6 Soft-decision list decoding of Reed-Solomon code (63, 31)

Designed output length l	Number of polynomials $l + 1$	Soft-decision		Hard-decision	
		C_M	$\frac{2}{3}(C_M+1)^3$	C_m	$\frac{2}{3}(C_m+1)^3$
2	3	84	4.09×10^5	63	1.75×10^5
4	5	211	6.35×10^6	189	4.75×10^6
8	9	631	1.68×10^8	630	1.67×10^8
13	14	1470	2.12×10^9	1323	1.55×10^9
55	56	22371	7.46×10^{12}	22113	7.21×10^{12}

Table 5.1 Decoding complexity comparison for soft-decision and hard-decision list decoding of Reed-Solomon code (63, 15)

Designed output length l	Number of polynomials $l + 1$	Soft-decision		Hard-decision	
		C_M	$\frac{2}{3}(C_M+1)^3$	C_m	$\frac{2}{3}(C_m+1)^3$
1	2	90	5.12×10^5	63	1.75×10^5
4	5	451	6.17×10^7	378	3.63×10^7
7	8	1082	8.48×10^8	945	5.64×10^8
19	20	6307	1.67×10^{11}	5733	1.26×10^{11}

Table 5.2 Decoding complexity comparison for soft-decision and hard-decision list decoding of Reed-Solomon code (63, 31)

Simulation result comparisons are made based on output length l because for output length l, there are $l + 1$ polynomials taking part in the iterative interpolation process. The total number of iterations (C_m = (4.12) for hard-decision. C_M (5.11) for soft-decision) also grow with length l. Both the number of polynomials $l + 1$ and the number of iterations (C_m, C_M) are the important parameters that determine the decoding complexity. As mentioned in section 4.5, by knowing the total number of iterations, the list decoding system's decoding complexity can be approximately predicted by $\frac{2}{3}(C_m + 1)^3$ for hard-decision and $\frac{2}{3}(C_M + 1)^3$ for soft decision [52, 59]. Table 5.1 and 5.2 present the decoding complexity comparison between soft-decision and hard-decision based on the designed output length *l* for Reed-Solomon codes (63, 15) and (63, 31) respectively. From Table 5.1 and 5.2, it can be observed that based on the same value of *l*, soft-decision costs a bit higher decoding complexity than the hard-decision, but remains in the same order of decoding complexity. For example, list decoding of Reed-Solomon code (63, 15) with designed output length $l = 2$, soft-decision costs 4.09×10^5 finite field calculations while hard-decision costs 1.75×10^5 finite field calculations. However, significant changes of decoding complexity are still due to the changes of output length *l*. Notice that values C_M presented in the above two tables are the average values obtained from simulation observation. Even though soft-decision has higher decoding complexity than the hard-decision based on the same designed length, from Fig 5.5 and 5.6 it can be seen that soft-decision can achieved significant coding gain over the hard-decision, especially over Rayleigh fading channel. For example, over Rayleigh fading channel and based on designed length $l = 2$, soft-decision list decoding of Reed-Solomon code (63, 15) can achieve about 5.8 dB coding gain at BER = 10^{-5} compared with hard-decision. The performance improvement of soft-decision list decoding over hard-decision list decoding is achieved by insignificant complexity penalty is different to other types of coding schemes for which soft-decision decoding does increase decoding complexity significantly. This is because for the list decoding algorithm, the complexity is mainly dominated by the interpolation process, compared with which the complexity introduced by the priori process (Algorithm 5.1) is marginal. And for the interpolation process, the important parameter that determines its complexity is the iteration number. As the iteration number of soft-decision does not vary much from hard-decision based on the same designed output length, the complexity of the soft-decision list decoding is not much higher than the hard-decision.

It is important to point out that the hard-decision scheme with large designed output length costs very high decoding complexity, as indicated by Table 5.1 and 5.2. However, by using the soft-decision scheme with small output length can outperform hard-decision scheme's complexity expensive results. For example, hard-decision list decode Reed-Solomon code (63, 31) with designed length l = 19, referring to Table 5.2, the decoding complexity is approximately 1.26×10^{11}. However, using soft-decision scheme with output length l = 1 can already achieve better performance over Rayleigh fading channel. The decoding complexity for l = 1 is only approximately 5.12×10^5 which is much lower than the hard-decision.

It is also worthy to mention that the soft-decision scheme's optimal result is obtained by using equation (5.21) without running through interpolation and factorisation processes. Assume that the decoder acknowledges the transmitted code word symbols $c_0, c_1, \ldots, c_{n-1}$, after the reliability matrix Π has been obtained, equation (5.21) can be used to assess the soft-decision scheme's optimal result. If equation (5.21) is satisfied, decoding is claimed to be successful. Otherwise, decoding fails. From Figs 5.5 and 5.6, it can be seen that soft-decision scheme approaches its optimal result with the designed length of output list increases.

5.8 Conclusion

The chapter presented the soft-decision list decoding scheme for Reed-Solomon codes. At the receiver, the received word's reliability information is obtained. The information is then converted into multiplicity information based on which the interpolation process is performed. It was shown in the chapter how to obtain the reliability values and how to convert them into multiplicity values. For the algorithm that converts reliability values into multiplicity values, a practical method to realise the stopping rule based on the designed length of output list was introduced. Applying the complexity reduction scheme for interpolation, the soft-decision scheme's interpolation complexity can also be reduced based on knowing the iteration number. Simulation results show that based on the same designed length of output list, the soft-decision scheme has significant coding gains compared with the hard-decision scheme, but is at the higher expense of decoding complexity. It is also shown that the soft-decision scheme with a small output list length and lower decoding complexity can outperform the hard-decision scheme's optimal result which is not suitable for practical implementation due to a very high decoding complexity.

Chapter 6 Hard-Decision List Decoding of Hermitian Codes

6.1 Introduction

This chapter presents the hard-decision list decoding algorithm which is called the Guruswami-Sudan (GS) algorithm for one of the best performing algebraic-geometric codes – Hermitian codes. Performance of Hermitian codes by using the conventional unique decoding algorithm – Sakata algorithm with majority voting [5, 6, 25], has been investigated by Johnston and Carrasco [29, 31]. However, the unique decoding algorithm's error-correction capability is limited by the half distance bound $\left\lfloor \frac{d^* - 1}{2} \right\rfloor$ (3.22), where d^* is the designed minimum distance of the code. The GS algorithm can correct errors beyond this bound. This chapter presents the mathematical framework of the GS algorithm for its application to Hermitian codes, so as to engineer the decoding process. It consists of two processes: interpolation, to build an interpolated polynomial based on the received information and factorisation, to find the transmitted message information based on the interpolated polynomial. By first defining a Hermitian curve, these processes can be implemented with an iterative polynomial construction algorithm and a recursive coefficient search algorithm respectively. The first simulation results of GS decoding Hermitian codes was published by the author in [65]. However, list decoding of Hermitian codes with the GS algorithm remains complex and limits the application of the GS algorithm to longer codes. According to the complexity analysis in [52, 65], the GS algorithm's high complexity is mainly caused by the iterative interpolation, in which a group of polynomials are tested for different zero conditions and modified interactively. In section 3.5.3, to define the zero condition of a polynomial for Hermitian codes we need to transfer it into a polynomial written with respect to the zero basis functions of a Hermitian curve, which is not very efficient for implementation. However, the zero condition of a polynomial can also be defined without this transfer based on knowledge of the corresponding coefficients between the pole basis monomials and zero basis functions of a Hermitian curve. Inspired by this, a new algorithm to determine these coefficients is proposed in the chapter. These coefficients can be applied afterwards in the interpolation process. In order to improve the list decoding efficiency for Reed-Solomon codes, a complexity reduction scheme which identifies any unnecessary polynomials in the group and eliminates them during the iterative interpolation is proposed in [52] and described in Chapter 4 of the thesis. From this project's research, this scheme is also valid for list decoding of Hermitian codes. In this chapter, the modified interpolation process will be presented with applying this complexity reduction scheme. The complexity analysis of this modification scheme shows that it can reduce decoding complexity up to 48.83%. The above work on reducing interpolation complexity was written in the author's paper [66] which has been accepted for publication. The factorisation process can be implemented by applying the recursive coefficient search algorithm which was first introduced by Roth and Ruckenstein [10] with application for Reed-Solomon codes, and later extended by Wu and Siegel [11, 12] for general algebraic-geometric codes. A more general factorisation algorithm which can be applied for both Reed-Solomon and algebraic-geometric codes is presented by the author in [67]. Based on the complexity reduction interpolation process and the more general factorisation process, list decoding results for longer Hermitian codes have been achieved. This chapter presents simulation results for the list decoding of Hermitian codes with comparisons to the unique decoding algorithm - the Sakata algorithm with majority voting [5, 6, 25]. Also, a comparison of Hermitian codes and Reed-Solomon codes using the list decoding algorithm is presented.

6.2 Prerequisite Knowledge

Here gives a short review for the prerequisite knowledge of Hermitian codes which was mentioned in Chapter 3. A Hermitian curve defined over GF(q) is given as:

$$H_w(x, y) = x^{w+1} + y^w + y \tag{6.1}$$

where $w = \sqrt{q}$ and has a genus $g = \frac{w(w-1)}{2}$ [19]. For simplicity, GF(q) is assumed to be an extension field of GF(2). There are $n = w^3$ affine points $p_i = (x_i, y_i)$ that satisfy $H_w(x_i, y_i) = 0$ and a point at infinity p_∞. On curve H_w, the pole order at p_∞ (v_{p_∞}) of variable x and y are $v_{p_\infty}(x^{-1}) = w$ and $v_{p_\infty}(y^{-1}) = w + 1$ [19]. With respect to the point at infinity p_∞, there exists a pole basis L_w which contains a set of bivariate monomials $\phi_a(x, y)$ with coefficients 1 and increasing pole orders, defined by (3.23) as: $L_w = \{\phi_a(x, y) \mid v_{p_\infty}(\phi_a^{-1}) < v_{p_\infty}(\phi_{a+1}^{-1}), a \in \mathrm{N}\}$ [9, 44], where the x degree of ϕ_a is not greater than w and N is the set of nonnegative integers. L_w defines the set of pole basis functions of the Hermitian curve H_w. A couple of examples of L_w is given in section 3.4. Nonnegative integers can be divided into nongaps which are the pole orders of monomials in L_w, and gaps otherwise. Take $L_4 = \{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, y^4, x^4y, x^3y^2, x^2y^3, xy^4, y^5, \ldots\}$ shown by example 3.2 for analysis. In L_4, nonnegative integers 1, 2, 3, 6, 7, and 11 are gaps while the rest of the nonnegative integers are nongaps. With respect to every affine point p_i, there exists a zero basis Z_{w,p_i} which contains a set of rational functions $\psi_{p_i,\alpha}(x, y)$ with increasing zero orders at p_i (v_{p_i}), defined by (3.24) as: $Z_{w,p_i} = \{\psi_{p_i,\alpha}(x, y) \mid v_{p_i}(\psi_{p_i,\alpha}) < v_{p_i}(\psi_{p_i,\alpha+1}), \alpha \in \mathrm{N}\}$ [9, 44]. $\psi_{p_i,\alpha}$ has a zero order α at affine point p_i. According to (3.25), it can be generally written as:

$$\psi_{p_i,\alpha}(x, y) = \psi_{p_i,\lambda+(w+1)\delta}(x, y) = (x - x_i)^{\lambda}[(y - y_i) - x_i^{w}(x - x_i)]^{\delta}, (\lambda, \delta \in \mathrm{N}, 0 \le \lambda \le w, \delta \ge 0)$$

The relationship between pole basis monomial ϕ_a and zero basis function $\psi_{p_i,\alpha}$ can be written as [44]:

$$\phi_a = \sum_{\alpha \in \mathrm{N}} \gamma_{a,p_i,\alpha} \psi_{p_i,\alpha} \tag{6.2}$$

where $\gamma_{a,p_i,\alpha} \in \mathrm{GF}(q)$ are the corresponding coefficients.

The construction of a (n, k) Hermitian code can be described as evaluating the n affine points of H_w over the message polynomial f, which is shown by equation (3.20) and (3.21) in Chapter 3.

To decode a (n, k) Hermitian code with the GS algorithm, the pole order of variable z is defined as $w_z = v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{k-1}^{-1})$ and $w_z > 2g - 1$. Then, any trivariate monomial $\phi_a z^b$'s (1, w_z)-weighted degree can be defined as:

$$\deg_{1,w_z}(\phi_a z^b) = v_{p_\infty}(\phi_a^{-1}) + b \cdot w_z \tag{6.3}$$

and a (1, w_z)-lexicographic order (ord) can be defined to arrange monomials $\phi_a z^b$:

$$\phi_{a_1} z^{b_1} < \phi_{a_2} z^{b_2}$$

if $\deg_{1,w_z}(\phi_{a_1} z^{b_1}) < \deg_{1,w_z}(\phi_{a_2} z^{b_2})$, or $\deg_{1,w_z}(\phi_{a_1} z^{b_1}) = \deg_{1,w_z}(\phi_{a_2} z^{b_2})$ and $b_1 < b_2$ [44]. For example, to list decode Hermitian code (8, 4, 4) which is defined in GF(4), $w_z = 4$. The (1, 4)-weighted degree and (1, 4)-lexicographic order of monomial $\phi_a z^b$ ($\phi_a \in L_2$) are shown in Table 6.1a and 6.1b respectively.

a b	0	1	2	3	4	5	6	7	8	9	10	11	……
0	0	2	3	4	5	6	7	8	9	10	11	12	……
1	4	6	7	8	9	10	11	12	……				
2	8	10	11	12	……								
3	12	……											
⋮	⋮												

Table 6.1a (1, 4)-weighted degree of monomial $\phi_a z^b$ ($\phi_a \in L_2$)

a b	0	1	2	3	4	5	6	7	8	9	10	11	……
0	0	1	2	3	5	6	8	10	13	15	18	21	……
1	4	7	9	11	14	16	19	22	……				
2	12	17	20	23	……								
3	24	……											
⋮	⋮												

Table 6.1b (1, 4)-lexicographic order of monomial $\phi_a z^b$ ($\phi_a \in L_2$)

$F_q[x, y, z]$ is the ring of polynomials defined over the set of pole basis functions in L_w of the Hermitian curve H_w, which can be generally written as: $f(x, y, z) = \sum_{a,b \in \mathrm{N}} f_{ab}\phi_a(x, y)z^b$, where $f_{ab} \in \mathrm{GF}(q)$ and $\phi_a \in L_w$. Subsequently, $F_q[x, y]$ is a subset of $F_q[x, y, z]$ with z degree equals to 0 and $F_q^u[x, y]$ is a subset of $F_q[x, y]$ with $v_{p_\infty}(\phi_a^{-1}) \le u$. As $w_z = v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{k-1}^{-1})$, the message polynomial (3.20) is a polynomial in $F_q^{w_z}[x, y]$. The following definition is given for polynomials defined in $F_q[x, y, z]$:

Definition 1: If $\phi_{a'} z^{b'}$ is the maximal monomial in polynomial $f \in F_q[x, y, z]$ as:

$$\phi_{a'}z^{b'} = \max\{\phi_a z^b \mid f_{ab} \neq 0\}$$

$\phi_{a'}z^{b'}$ is called f's leading monomial and its coefficient $f_{a'b'}$ is called f's leading coefficient, denoted as:

$$\text{LM}(f) = \phi_{a'}z^{b'}, \text{ and } \text{LC}(f) = f_{a'b'}$$

and polynomial f's $(1, w_z)$-weighted degree ($\deg_{1,w_z}(f)$) and leading order (lod(f)) are defined as:

$$\deg_{1,w_z}(f) = \deg_{1,w_z}(\phi_{a'}z^{b'}), \text{ and } \text{lod}(f) = \text{ord}(\phi_{a'}z^{b'})$$

For example, f is a polynomial in $F_4[x, y, z]$ and can be written as: $f(x, y, z) = \sigma x + \sigma^2x^2 + \sigma xy + \sigma^2xz + yz^2 + \sigma z^3 = \sigma \cdot \phi_1z^0 + \sigma^2 \cdot \phi_3z^0 + \sigma \cdot \phi_4z^0 + \sigma^2 \cdot \phi_1z^1 + 1 \cdot \phi_2z^2 + \sigma \cdot \phi_0z^3$, where σ is a primitive element in GF(4) satisfying $\sigma^2 + \sigma + 1 = 0$. Applying the (1, 4)-lexicographic order shown by Table 6.1b, it can be seen that the leading monomial of f is $\phi_{a'}z^{b'} = \phi_0z^3$. Therefore, LM($f$) = ϕ_0z^3, LC(f) = σ, $\deg_{1,4}(f) = \deg_{1,4}(\phi_0z^3) = 12$ and lod(f) = ord(ϕ_0z^3) = 24.

Based on the above definition, for any two polynomials f and $h \in F_q[x, y, z]$, $f < h$ if lod(f) < lod(h).

6.3 GS Decoding of Hermitian Codes

The GS algorithm consists of two processes: interpolation and factorisation. Given a received word $R = (r_0, r_1, \ldots, r_{n-1})$ ($r_i \in \text{GF}(q)$, $i = 0, 1, \ldots, n - 1$), n interpolated units can be formed by combining each received symbol with its respective affine point used in encoding as: (p_0, r_0), (p_1, r_1), …, (p_{n-1}, r_{n-1}). Interpolation is to build the minimal polynomial $Q \in F_q[x, y, z]$ which has a zero of multiplicity at least m over the n interpolated units. In general, Q can be written as (3.30): $Q = \sum_{a,b \in \mathbb{N}} Q_{ab}\phi_a z^b$, where $Q_{ab} \in \text{GF}(q)$ and $\phi_a \in L_w$. If (p_i, r_i) is the intended interpolated unit, it can also be written with respect to the zero basis functions in Z_{w,p_i} as [44]:

$$Q = \sum_{\alpha,\beta\in\mathrm{N}} Q_{\alpha\beta}^{(p_i,r_i)} \psi_{p_i,\alpha} (z-r_i)^{\beta} \tag{6.4}$$

where $Q_{\alpha\beta}^{(p_i,r_i)} \in$ GF(q). If $Q_{\alpha\beta}^{(p_i,r_i)} = 0$ for $\alpha + \beta < m$, polynomial Q has a zero of multiplicity at least m at unit (p_i, r_i) [9, 44]. As $z^b = (z - r_i + r_i)^b = \sum_{\beta\le b}\binom{b}{\beta} r_i^{b-\beta}(z-r_i)^{\beta}$ and $\phi_a = \sum_{\alpha\in\mathrm{N}} \gamma_{a,p_i,\alpha}\psi_{p_i,\alpha}$, substitute them into (3.30) as:

$$Q = \sum_{a,b\in\mathrm{N}} Q_{ab}\Big(\sum_{\alpha\in\mathrm{N}} \gamma_{a,p_i,\alpha}\psi_{p_i,\alpha}\Big)\Big(\sum_{\beta\le b}\binom{b}{\beta} r_i^{b-\beta}(z-r_i)^{\beta}\Big)$$

$$= \sum_{\alpha,\beta\in\mathrm{N}}\Big(\sum_{a,b\ge\beta} Q_{ab}\binom{b}{\beta}\gamma_{a,p_i,\alpha} r_i^{b-\beta}\Big)\psi_{p_i,\alpha}(z-r_i)^{\beta} \tag{6.5}$$

Therefore, coefficients $Q_{\alpha\beta}^{(p_i,r_i)}$ of (6.4) can be written as:

$$Q_{\alpha\beta}^{(p_i,r_i)} = \sum_{a,b\ge\beta} Q_{ab}\binom{b}{\beta}\gamma_{a,p_i,\alpha} r_i^{b-\beta} \tag{6.6}$$

(6.6) defines the zero condition constraints to the coefficients Q_{ab} of polynomial Q, so that Q has a zero of multiplicity at least m over unit (p_i, r_i). Here gives an example to show how to define the zero condition of a polynomial in $F_q[x, y, z]$ using (6.6).

Example 6.1 Given polynomial $Q(x, y, z) = 1 + \sigma y + \sigma x^2 + z^2(1 + \sigma^2 y)$ defined in $F_4[x, y, z]$. Justify it has a zero of multiplicity at least 2 over unit $(p, r) = ((1, \sigma), \sigma)$. σ is a primitive element in GF(4) satisfying $\sigma^2 + \sigma + 1 = 0$. Addition and multiplication tables of GF(4) is given in Appendix A.

Polynomial $Q(x, y, z) = 1 + \sigma y + \sigma x^2 + z^2(1 + \sigma^2 y) = Q_{00}\phi_0 z^0 + Q_{20}\phi_2 z^0 + Q_{30}\phi_3 z^0 + Q_{02}\phi_0 z^2 + Q_{22}\phi_2 z^2$. For supporting the zero condition calculations, the corresponding coefficients $\gamma_{a,p,\alpha}$ are shown in Table 6.2 as:

α \ a	0	1	2	3	…
0	1	1	σ	1	…
1	0	1	1	0	…
⋮	⋮	⋮	⋮	⋮	

Table 6.2 Corresponding coefficients $\gamma_{a,p,\alpha}$ given $p = (1, \sigma)$

Based on the above description, to justify Q has a zero of multiplicity m over unit (p, r), its coefficients Q_{ab} should satisfy $Q_{\alpha\beta}^{(p,r)} = 0$ for $\alpha + \beta < 2$ as: $Q_{00}^{(p,r)} = 0$, $Q_{01}^{(p,r)} = 0$ and $Q_{10}^{(p,r)} = 0$.

Based on definition (6.6),

$$Q_{00}^{(p,r)} = Q_{00}\binom{0}{0}\gamma_{0,p,0}\sigma^{0\text{-}0} + Q_{20}\binom{0}{0}\gamma_{2,p,0}\sigma^{0\text{-}0} + Q_{30}\binom{0}{0}\gamma_{3,\text{p},0}\sigma^{0\text{-}0} + Q_{02}\binom{2}{0}\gamma_{0,p,0}\sigma^{2\text{-}0} + Q_{22}\binom{2}{0}\gamma_{2,p,0}\sigma^{2\text{-}0} = 1 + \sigma^2 + \sigma + \sigma^2 + \sigma^2 = 0$$

$$Q_{01}^{(p,r)} = Q_{02}\binom{2}{1}\gamma_{0,p,0}\sigma^{2\text{-}1} + Q_{22}\binom{2}{1}\gamma_{2,p,0}\sigma^{2\text{-}1} = 0 + 0 = 0$$

$$Q_{10}^{(p,r)} = Q_{00}\binom{0}{0}\gamma_{0,p,1}\sigma^{0\text{-}0} + Q_{20}\binom{0}{0}\gamma_{2,p,1}\sigma^{0\text{-}0} + Q_{30}\binom{0}{0}\gamma_{3,p,1}\sigma^{0\text{-}0} + Q_{02}\binom{2}{0}\gamma_{0,p,1}\sigma^{2\text{-}0} + Q_{22}\binom{2}{0}\gamma_{2,p,1}\sigma^{2\text{-}0} = \sigma + \sigma = 0.$$

Therefore, polynomial Q has a zero of multiplicity at least 2 over unit (p, r) = ((1, σ), σ).

If constraint (6.6) for the coefficients of polynomial Q is denoted as $D_{\alpha\beta}^{(p_i,r_i)}(Q)$, such that:

$$D_{\alpha\beta}^{(p_i,r_i)}(Q) = Q_{\alpha\beta}^{(p_i,r_i)} = \sum_{a,b\geq\beta} Q_{ab}\binom{b}{\beta}\gamma_{a,p_i,\alpha}r_i^{b-\beta} \qquad (6.7)$$

then interpolation is to build a polynomial Q defined as:

$$Q = \min_{lod(Q)} \{Q \in F_q[x, y, z] \mid D_{\alpha\beta}^{(p_i, r_i)}(Q) = 0 \text{ for } i = 0, 1, \ldots, n-1 \wedge \alpha + \beta < m \ (\alpha, \beta \in \mathrm{N})\} \tag{6.8}$$

As there are $\begin{pmatrix} m+1 \\ 2 \end{pmatrix}$ permutations of (α, β) for $\alpha + \beta < m$, there are in total:

$$C = n\begin{pmatrix} m+1 \\ 2 \end{pmatrix} \tag{6.9}$$

zero condition constraints that coefficients Q_{ab} of polynomial Q need to satisfy. C also represents the number of iterations in the interpolation algorithm [9, 44], in which each iteration imposes a zero condition constraint to Q_{ab}. The (1, w_z)-weighted degree upper bound of polynomial Q is defined as [9, 44]:

$$\max\{\deg_{1,w_z} Q\} = l_m \ v_{p_\infty}(z^{-1}) + t_m \tag{6.10}$$

where l_m is the maximal number of output candidates from factorisation, defined as:

$$l_m = \max\{u \mid \begin{pmatrix} u \\ 2 \end{pmatrix} v_{p_\infty}(z^{-1}) - (u - 1)\, g \le C\} - 1 \tag{6.11}$$

and parameter t_m is defined as:

$$t_m = \max\{u \mid (l_m + 1)\, u - \Gamma(u) + \begin{pmatrix} l_m + 1 \\ 2 \end{pmatrix} v_{p_\infty}(z^{-1}) - l_m g \le C\} \tag{6.12}$$

where $u \in \mathrm{N}$ and $\Gamma(u)$ denotes the number of gaps that are less than or equal to the nonnegative integer u [9].

If there exists a polynomial $h \in F_q^{u_z}[x, y]$ such that

$$\Lambda(h, R) = |\{i \mid h(p_i) = r_i, i = 0, 1, \ldots, n - 1\}| \tag{6.13}$$

the total zero orders of polynomial $Q(x, y, h)$ over all the interpolated units is:

$$\sum_{i=0}^{n-1} v_{p_i}(Q(x, y, h)) \ge m\, \Lambda(h, R) \tag{6.14}$$

To define the total zero order of polynomial $Q(x, y, h)$, the following lemma is applied.

Lemma 6.1 $Q(x, y, z)$ has a zero of multiplicity m over unit (p_i, r_i) and h is a polynomial in $F_q^{u_z}[x, y]$ that satisfies $h(p_i) = r_i$, then $Q(x, y, h)$ has a zero order at least m at p_i, as $v_{p_i}(Q(x, y, h)) \geq m$ [9, 44].

Equation (6.13) defines the total number of affine points that satisfy $h(p_i) = r_i$, and therefore the total zero order of polynomial $Q(x, y, h)$ over all the affine points is defined by equation (6.14).

Theorem 6.2 If polynomial $Q(x, y, h)$'s total zero orders is greater than its pole order as:

$$\sum_{i=0}^{n-1} v_{p_i}(Q(x, y, h)) > v_{p_\infty}(Q(x, y, h)^{-1}) \tag{6.15}$$

then h is the z root of Q: $Q(x, y, h) = 0$, or equivalently $z - h \mid Q(x, y, z)$ [7, 9, 44].

As $h \in F_q^{u_z}[x, y]$, $v_{p_\infty}(Q(x, y, h)^{-1}) = v_{p_\infty}(Q(x, y, z)^{-1}) = \deg_{1,w_z}(Q(x, y, z))$. Therefore, based on (6.13) and (6.14), theorem 6.2 results the following corollary:

Corollary 6.3: If there exists a polynomial $h \in F_q^{u_z}[x, y]$ such that:

$$m\,\Lambda(h, R) > \deg_{1,w_z}(Q(x, y, z)) \tag{6.16}$$

list decoding outputs h can be found out by factorising the interpolated polynomial $Q(x, y, z)$ as: $z - h \mid Q(x, y, z)$.

Factorisation is to find the z roots of the interpolated polynomial Q, among which the message polynomial (3.20) is included [11, 12, 65]. If $h = f$, equation (6.13) defines the number of uncorrupted received symbols. Therefore, the GS algorithm's error-correction capability τ_m is:

$$\tau_m = n - \Lambda(h, R) = n - \left\lfloor \frac{\deg_{1,w_z} Q}{m} \right\rfloor - 1 \tag{6.17}$$

As the upper bound of $\deg_{1,w_z} Q$ is defined by (6.10), therefore:

$$\tau_m \geq n - \left\lfloor \frac{l_m v_{p_\infty}(z^{-1}) + t_m}{m} \right\rfloor - 1 \tag{6.18}$$

According to the theoretical background description given in Chapter 3, the GS algorithm's error-correction capability upper bound for a (n, k) Hermitian code is defined by equation (3.32) as: $\tau_{GS} = n - \lfloor \sqrt{n(n-d^*)} \rfloor - 1$.

6.4 Determining the Corresponding Coefficients

Based on equation (6.7), the corresponding coefficients $\gamma_{a,p_i,\alpha}$ are critical for defining the zero condition of a polynomial in $F_q[x, y, z]$. Without knowing them, we have to transfer a general polynomial (3.30) into (6.4) and find the coefficients $Q_{\alpha\beta}^{(p_i,r_i)}$, which is not efficient during the iterative interpolation. In fact, the corresponding coefficients $\gamma_{a,p_i,\alpha}$ can be determined independently of the received word. And therefore, if they can be determined beforehand and applied during the iterations, the interpolation efficiency can be greatly improved. This section proposes an algorithm to determine them.

The problem we intend to solve can be simply stated as: given an affine point $p_i = (x_i, y_i)$ of curve H_w and a pole basis monomial ϕ_a, determine the corresponding coefficients $\gamma_{a,p_i,\alpha}$ so that ϕ_a can be written as a sum of the zero basis functions $\psi_{p_i,\alpha}$: $\phi_a = \sum_{\alpha \in \mathrm{N}} \gamma_{a,p_i,\alpha} \psi_{p_i,\alpha}$. For any two pole basis monomials ϕ_{a_1} and ϕ_{a_2} in L_w, ϕ_{a_1} ϕ_{a_2} = $\sum_{a \in \mathrm{N}} \phi_a$ and the zero basis function $\psi_{p_i,\alpha}$ (3.25) can be written as a sum of pole basis monomials ϕ_a as [44]:

$$\psi_{p_i,\alpha} = \sum_{a \in \mathrm{N}} \zeta_a \phi_a \tag{6.19}$$

where coefficients $\zeta_a \in \mathrm{GF}(q)$. Based on (3.25), partition $\psi_{p_i,\alpha}(x, y)$ as:

$$\psi_{p_i,\alpha} = \psi^A_{p_i,\alpha} \cdot \psi^B_{p_i,\alpha} \tag{6.20}$$

where $\psi^A_{p_i,\alpha} = (x - x_i)^\lambda$ and $\psi^B_{p_i,\alpha} = [(y - y_i) - x_i^w(x - x_i)]^\delta = [y - x_i^w x - (y_i - x_i^{w+1})]^\delta$. It is easy to recognise that $\psi^A_{p_i,\alpha}$ has leading monomial LM($\psi^A_{p_i,\alpha}$) = x^λ and leading coefficient LC($\psi^A_{p_i,\alpha}$) = 1. As $v_{p_\infty}(y^{-1}) > v_{p_\infty}(x^{-1})$, $\psi^B_{p_i,\alpha}$ has leading monomial LM($\psi^B_{p_i,\alpha}$) = y^δ and leading coefficient LC($\psi^B_{p_i,\alpha}$) = 1. Based on (6.20), $\psi_{p_i,\alpha}$ has leading monomial LM($\psi^A_{p_i,\alpha}$) · LM($\psi^B_{p_i,\alpha}$) = $x^\lambda y^\delta$ and leading coefficient LC($\psi^A_{p_i,\alpha}$) · LC($\psi^B_{p_i,\alpha}$) = 1. As $0 \leq \lambda \leq w$ and $\delta \geq 0$, the set of leading monomials of zero basis functions in Z_{w,p_i} contains all the monomials defined in pole basis L_w. Summarising the above analysis, corollary 6.4 is proposed as followed.

Corollary 6.4: If ϕ_L is the leading monomial of zero basis function $\psi_{p_i,\alpha}$ as LM($\psi_{p_i,\alpha}$) = ϕ_L, the leading coefficient of $\psi_{p_i,\alpha}$ equals to 1 and (6.19) can be written as:

$$\psi_{p_i,\alpha} = \sum_{a \in \mathrm{N}, a < L} \zeta_a \phi_a + \phi_L \tag{6.21}$$

The set of leading monomials of zero basis functions in Z_{w,p_i} contains all the monomials in L_w:

$$\{\mathrm{LM}(\psi_{p_i,\alpha}) = \phi_L,\ \psi_{p_i,\alpha} \in Z_{w,p_i}\} \subseteq L_w \tag{6.22}$$

Following on, by identifying the second largest pole basis monomial ϕ_{L-1} with coefficient $\zeta_{L-1} \in \mathrm{GF}(q)$ in $\psi_{p_i,\alpha}$, (6.21) can also be written as:

$$\psi_{p_i,\alpha} = \sum_{a \in \mathrm{N}, a < L-1} \zeta_a \phi_a + \zeta_{L-1}\phi_{L-1} + \phi_L \tag{6.23}$$

Now it is sufficient to propose the new efficient algorithm to determine the corresponding coefficients $\gamma_{a,p_i,\alpha}$.

Algorithm 6.1: Determine the corresponding coefficients $\gamma_{a,p_i,\alpha}$ between a pole basis monomial ϕ_a and zero basis functions $\psi_{p_i,\alpha}$.

```
(i) Initialise all corresponding coefficients $\gamma_{a,p_i,\alpha} = 0$
(ii) Find the zero basis function $\psi_{p_i,\alpha}$ with LM($\psi_{p_i,\alpha}$) = $\phi_a$, and let $\gamma_{a,p_i,\alpha} = 1$
(iii) Initialise function $\hat{\psi} = \psi_{p_i,\alpha}$
(iv) While ($\hat{\psi} \neq \phi_a$) {
(v)     Find the second largest pole basis monomial $\phi_{L-1}$ with coefficient $\zeta_{L-1}$ in $\hat{\psi}$
(vi)    In $Z_{w,p_i}$, find a zero basis function $\psi_{p_i,\alpha}$ whose leading monomial LM($\psi_{p_i,\alpha}$) = $\phi_{L-1}$, and let the corresponding coefficient $\gamma_{a,p_i,\alpha} = \zeta_{L-1}$
(vii)   Update $\hat{\psi} = \hat{\psi} + \gamma_{a,p_i,\alpha}\psi_{p_i,\alpha}$
}
```

Proof: Notice that functions $\psi_{p_i,\alpha}$ with LM($\psi_{p_i,\alpha}$) > ϕ_a will not contribute to the sum calculation of (6.2) and their corresponding coefficients $\gamma_{a,p_i,\alpha} = 0$. The zero basis function $\psi_{p_i,\alpha}$ found at (ii) has leading monomial $\phi_L = \phi_a$. Based on (6.23), it can be written as:

$$\psi_{p_i,\alpha} = \sum_{a'\in \mathrm{N}, a'<L-1} \zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} + \phi_a \tag{6.24}$$

(6.24) indicates the corresponding coefficient between ϕ_a and $\psi_{p_i,\alpha}$ is 1: $\gamma_{a,p_i,\alpha} = 1$. Polynomial $\hat{\psi}$ initialised by (iii) is an accumulated polynomial resulting in ϕ_a. While $\hat{\psi} \neq \phi_a$, in (6.24), the second largest monomial ϕ_{L-1} with coefficient ζ_{L-1} is identified by (v). Then, find another zero basis function $\psi_{p_i,\alpha}$ in Z_{w,p_i} that LM($\psi_{p_i,\alpha}$) = ϕ_{L-1}. According to corollary 6.4, this zero basis function always exists and it can be written as: $\psi_{p_i,\alpha} = \sum_{a'\in \mathrm{N}, a'<L-1} \zeta_{a'}\phi_{a'} + \phi_{L-1}$. At (vi), the corresponding coefficient between

monomial ϕ_a and the found zero basis function $\psi_{p_i,\alpha}$ can be determined as: $\gamma_{a,p_i,\alpha}$ = $\zeta_{L\text{-}1}$. As a result, the accumulated calculation of (vii) can be written as:

$$\hat{\psi} = \sum_{a'\in\mathrm{N},a'<L-1} \zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} + \phi_a + \gamma_{a,p_i,\alpha}\,\psi_{p_i,\alpha}$$

$$= \sum_{a'\in\mathrm{N},a'<L-1} \zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} + \phi_a + \sum_{a'\in\mathrm{N},a'<L-1} \zeta_{L-1}\zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} \quad (6.25)$$

Therefore in the new accumulated $\hat{\psi}$, $\zeta_{L\text{-}1}\phi_{L\text{-}1}$ is eliminated while the leading monomial ϕ_a is preserved. If the updated $\hat{\psi} \neq \phi_a$, its second largest monomial $\phi_{L\text{-}1}$ is again eliminated while ϕ_a is always preserved as a leading monomial by the same process. The algorithm terminates after all monomials that are smaller than ϕ_a have been eliminated and results in $\hat{\psi} = \phi_a$. This process is equivalent to the sum calculation of (6.2). Here a worked example is presented to illustrate algorithm 6.1.

Example 6.2: Given $p_i = (\sigma^2, \sigma^2)$ is an affine point on curve H_2 and a pole basis (L_2) monomial $\phi_5 = y^2$, determine the corresponding coefficients $\gamma_{5,p_i,\alpha}$ so that ϕ_5 can be written as $\phi_5 = \sum_{\alpha\in\mathrm{N}} \gamma_{5,p_i,\alpha}\psi_{p_i,\alpha}$.

Based on (3.25), the first 8 zero basis functions in Z_{2,p_i} can be listed as:

$\psi_{p_i,0} = (x - \sigma^2)^0 = 1 \qquad \psi_{p_i,1} = (x - \sigma^2)^1 = \sigma^2 + x$

$\psi_{p_i,2} = (x - \sigma^2)^2 = \sigma + x^2 \qquad \psi_{p_i,3} = (y - \sigma^2) - \sigma(x - \sigma^2) = \sigma + \sigma x + y$

$\psi_{p_i,4} = (x - \sigma^2)[(y - \sigma^2) - \sigma(x - \sigma^2)] = 1 + \sigma^2 x + \sigma^2 y + \sigma x^2 + xy$

$\psi_{p_i,5} = (x - \sigma^2)^2[(y - \sigma^2) - \sigma(x - \sigma^2)] = \sigma^2 + \sigma^2 x + \sigma x^2 + \sigma y^2 + x^2 y$

$\psi_{p_i,6} = [(y - \sigma^2) - \sigma(x - \sigma^2)]^2 = \sigma^2 + \sigma^2 x^2 + y^2$

$\psi_{p_i,7} = (x - \sigma^2)[(y - \sigma^2) - \sigma(x - \sigma^2)]^2 = \sigma + \sigma^2 x + \sigma^2 y + \sigma x^2 + xy^2.$

Initialise all $\gamma_{5,p_i,\alpha} = 0$. In Z_{2,p_i} , as LM($\psi_{p_i,6}$) = ϕ_5, we let $\gamma_{5,p_i,6} = 1$ and initialise the accumulated polynomial $\hat{\psi} = \psi_{p_i,6} = \sigma^2 + \sigma^2 x^2 + y^2$.

As $\hat{\psi} \neq \phi_5$, its second largest monomial $\phi_{L\text{-}1} = x^2$ with coefficient $\zeta_{L\text{-}1} = \sigma^2$ are identified. Among the zero basis functions in Z_{2,p_i}, we find $\psi_{p_i,2}$ with LM($\psi_{p_i,2}$) = $\phi_{L\text{-}1} = x^2$, and let $\gamma_{5,p_i,2} = \zeta_{L\text{-}1} = \sigma^2$. Update $\hat{\psi} = \hat{\psi} + \gamma_{5,p_i,2}\,\psi_{p_i,2} = \sigma + y^2$.

As $\hat{\psi} \neq \phi_5$, again its second largest monomial $\phi_{L\text{-}1} = 1$ with coefficient $\zeta_{L\text{-}1} = \sigma$ are identified. Among the zero basis functions in Z_{2,p_i}, we find $\psi_{p_i,0}$ with LM($\psi_{p_i,0}$) = $\phi_{L\text{-}1} = 1$, and let $\gamma_{5,p_i,0} = \zeta_{L\text{-}1} = \sigma$. Update $\hat{\psi} = \hat{\psi} + \gamma_{5,p_i,0}\,\psi_{p_i,0} = y^2$.

Now, $\hat{\psi} = \phi_5$, we can stop the algorithm and output $\gamma_{5,p_i,0} = \sigma$, $\gamma_{5,p_i,2} = \sigma^2$ and $\gamma_{5,p_i,6} = 1$. The rest of the corresponding coefficients $\gamma_{5,p_i,\alpha} = 0$ ($\alpha \neq 0, 2, 6$).

Before interpolation, monomials ϕ_a that exist in the interpolated polynomial Q are unknown. However, the (1, w_z)-weighted degree upper bound of polynomial Q is defined by (6.10), from which the largest pole basis monomial $\phi_{\max}$ that might exist in Q can be predicted by $v_{p_\infty}(\phi_{\max}^{-1}) = \max\{\deg_{1,w_z} Q\}$. Based on interpolation multiplicity m, with parameter $\alpha < m$, the corresponding coefficients that might be used in interpolation are $\gamma_{0,p_i,\alpha} \sim \gamma_{\max,p_i,\alpha}$ ($\alpha < m$). Therefore algorithm 6.1 can be used to determine all the corresponding coefficients $\gamma_{0,p_i,\alpha} \sim \gamma_{\max,p_i,\alpha}$ and only $\gamma_{0,p_i,\alpha} \sim \gamma_{\max,p_i,\alpha}$ ($\alpha < m$) are stored for interpolation in order to minimise the memory requirement. For example, to list decode the (8, 4, 4) Hermitian code with multiplicity $m = 2$, $\max\{\deg_{1,w_z} Q\} = 13$. Therefore, the largest pole basis monomial that might exist in Q is $\phi_{\max} = \phi_{12} = x^2y^3$ and algorithm 6.1 can be applied to calculate all the corresponding coefficients $\gamma_{0,p_i,\alpha} \sim \gamma_{12,p_i,\alpha}$ and $\gamma_{0,p_i,\alpha} \sim \gamma_{12,p_i,\alpha}$ ($\alpha < 2$) are stored.

6.5 Complexity reduction Interpolation

Interpolation is to determine polynomial Q defined by (6.8). This can be implemented by an iterative polynomial construction algorithm [9, 44, 46, 52]. At the beginning, a group of polynomials are initialised. During the iterations, they are tested by different zero condition constraints and modified interactively. As mentioned in section 6.3, there are in total C (6.9) iterations, after which the minimal polynomial in the group is chosen as the interpolated polynomial Q. According to the iterative process analysis given in Chapter 4 and also paper [52], the interpolated polynomial Q has leading order $\mathrm{lod}(Q) \leq C$. This indicates that those polynomials with leading order over C will not be the chosen candidates. Also, if there is a polynomial in the group with leading order over C during the iterations, the chosen polynomial Q has not been modified with this polynomial, otherwise $\mathrm{lod}(Q) > C$. Therefore, those polynomials with leading order greater than C can be eliminated from the group during iterations in order to save the unnecessary computations.

If $f \in F_q[x, y, z]$ has leading monomial $\mathrm{LM}(f) = \phi_{a'} z^{b'}$, polynomials in $F_q[x, y, z]$ can be partitioned into the following classes according to their leading monomial's z degree b' and $\phi_{a'}$'s pole order $v_{p_\infty}(\phi_{a'}^{-1})$ as:

$$V_{\lambda+w\delta} = \{f \in F_q[x, y, z] \mid b' = \delta \wedge v_{p_\infty}(\phi_{a'}^{-1}) = uw + \lambda, \mathrm{LM}(f) = \phi_{a'} z^{b'}, (\delta, u, \lambda) \in \mathrm{N}, \lambda < w\} \quad (6.26)$$

such that $F_q[x, y, z] = \bigcup_{\lambda,\delta\in\mathrm{N},\lambda<w} V_{\lambda+w\delta}$. According to section 6.3, the factorisation outputs are the z roots of Q. Therefore, the z degree of Q is less than or equal to the maximal number of the output list l_m (6.11) and Q is a polynomial chosen from the following classes:

$$V_j = V_{\lambda+w\delta} \qquad (0 \leq \lambda < w, 0 \leq \delta \leq l_m) \quad (6.27)$$

At the beginning of the iterative process, a group of polynomials are initialised to represent each of the polynomial classes defined by (6.27) as:

$$G = \{Q_j = Q_{\lambda+w\delta} = y^{\lambda} z^{\delta}, Q_j \in V_j\} \quad (6.28)$$

During the iterations, each polynomial Q_j in the group G is the minimal polynomial within its class V_j that satisfies all the tested zero conditions. At the beginning of each iteration, polynomial group G is modified by:

$$G = \{Q_j \mid \mathrm{lod}(Q_j) \leq C\} \quad (6.29)$$

in order to eliminate those polynomials with leading order over C. Then the remaining polynomials in G are tested by the zero condition constraint defined by (6.7) as:

$$\Delta_j = D_{\alpha\beta}^{(p_i, r_i)}(Q_j) \tag{6.30}$$

The determined corresponding coefficients $\gamma_{a,p_i,\alpha}$ are applied for this calculation. Those polynomials with $\Delta_j = 0$ satisfy the zero condition and do not need to be modified. However, those polynomials with $\Delta_j \neq 0$ need to be modified. Among them, find the index of the minimal polynomial as $j^{'}$ and record the minimal polynomial as $Q^{'}$:

$$j^{'} = \text{index}\,(\min_{lod(Q_j)} \{Q_j \mid \Delta_j \neq 0\}) \tag{6.31}$$

$$Q^{'} = Q_{j'} \tag{6.32}$$

For $Q_{j'}$, it is modified as:

$$Q_{j'} = (x - x_i)Q^{'} \tag{6.33}$$

where x_i is the x coordinate of affine point p_i which is included in the current interpolated unit (p_i, r_i). The modified $Q_{j'}$ satisfies $D_{\alpha\beta}^{(p_i, r_i)}(Q_{j'}) = 0$. Based on property 1 and 2 mentioned in section 4.3.2, $D_{\alpha\beta}^{(p_i, r_i)}[(x - x_i)Q^{'}] = D_{\alpha\beta}^{(p_i, r_i)}(xQ^{'}) - x_i D_{\alpha\beta}^{(p_i, r_i)}(Q^{'}) = x_i \Delta_{j'} - x_i\Delta_{j'} = 0$. The rest of the polynomials with $\Delta_j \neq 0$ are modified as:

$$Q_j = \Delta_{j'}Q_j - \Delta_j Q^{'} \tag{6.34}$$

The modified Q_j satisfies $D_{\alpha\beta}^{(p_i, r_i)}(Q_j) = 0$ because $D_{\alpha\beta}^{(p_i, r_i)}[\Delta_{j'}Q_j - \Delta_j Q^{'}] = \Delta_{j'} D_{\alpha\beta}^{(p_i, r_i)}(Q_j) - \Delta_j D_{\alpha\beta}^{(p_i, r_i)}(Q^{'}) = \Delta_{j'}\Delta_j - \Delta_j \Delta_{j'} = 0$. After C iterations, the minimal polynomial in the group G is chosen as the interpolated polynomial Q:

$$Q = \min_{lod(Q_j)} \{Q_j \mid Q_j \in G\} \tag{6.35}$$

From the above description, it can be seen that by applying the complexity reduction scheme (6.29), zero condition calculation (6.30) and modifications (6.33) (6.34) for those polynomials Q_j with lod(Q_j) > C can be avoided, and therefore the interpolation efficiency can be improved. According to [52] and Chapter 4, this complexity reduction scheme is error dependent that it can reduce complexity more significantly in low error weight situations. This is because the modification scheme (6.29) takes action in earlier iteration steps while in low error weight situations, and therefore more computation can be reduced. Fig. 6.1 shows interpolation (with different multiplicity *m*) complexity reduction by applying the scheme (6.29) for decoding Hermitian code (64, 19, 40). It is shown that complexity can be reduced significantly in low error weight situations, especially when *m* = 1, complexity can be reduced up to 48.83%. However, in high error weight situations, complexity reduction is not as significant. Based on Fig. 6.1, it can also be observed that the complexity reduction also depends on the interpolation multiplicity *m*. When *m* = 1, complexity reduction is the most significant; when *m* = 2, complexity reduction is the most marginal.

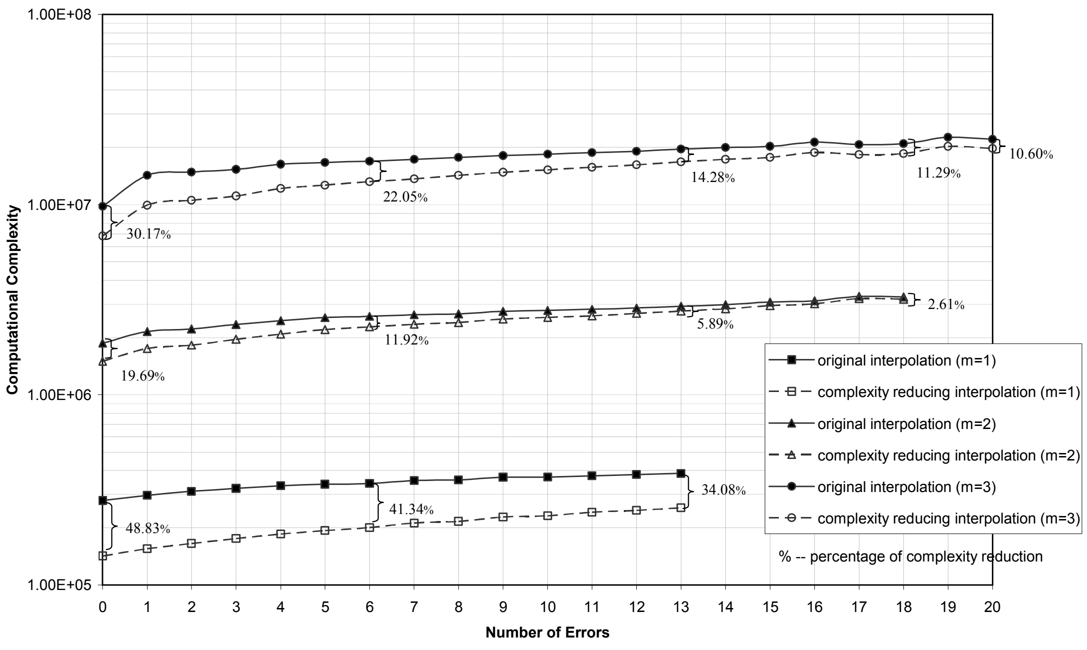


Figure 6.1 Complexity analysis for the interpolation of GS decoding Hermitian code (64, 19, 40)

Summarising section 6.4 and 6.5, the modified complexity reduction interpolation process for GS decoding Hermitian codes can be stated as:

Initial computation: Apply algorithm 6.1 to determine all the necessary corresponding coefficients $\gamma_{a,p_i,\alpha}$ and store them to be used by the iterative polynomial construction algorithm (algorithm 6.2)

Algorithm 6.2: Iterative Polynomial Construction

Initialisation: Initialise the group of polynomials *G* by (6.28)

```
(i): For each interpolated unit $(p_i, r_i)$ $(i = 0, 1, \ldots, n-1)$ {
(ii):     For each pair of the zero condition parameters $(\alpha, \beta)$ $(\alpha + \beta < m)$ {
(iii):    Modify polynomial group $G$ by (6.29)
(iv):     Test the zero condition $\Delta_j$ of each polynomial in $G$ by (6.30)
(v):      For polynomials $Q_j$ with $\Delta_j \neq 0$ {
(vi):         Denote the minimal polynomial's index as $j'$ by (6.31) and record it as $Q'$ by (6.32)
(vii):        If $j = j'$, $Q_j$ is modified by (6.33)
(viii):       If $j \neq j'$, $Q_j$ is modified by (6.34)
}}}
```

At the end of the iterations, the minimal polynomial Q is chosen from the group G as (6.35).

Here gives an example to illustrate this complexity reduction interpolation process.

Example 6.3 Decode Hermitian code (8, 4, 4) defined in GF(4) using the GS algorithm with interpolation multiplicity $m = 2$. The Hermitian code word is generated by evaluating the message polynomial over the following affine points: $p_0 = (0, 0)$, $p_1 = (0, 1)$, $p_2 = (1, \sigma)$, $p_3 = (1, \sigma^2)$, $p_4 = (\sigma, \sigma)$, $p_5 = (\sigma, \sigma^2)$, $p_6 = (\sigma^2, \sigma)$, $p_7 = (\sigma^2, \sigma^2)$. Given received word $R = (1, \sigma^2, \sigma, \sigma^2, \sigma, \sigma^2, \sigma^2, \sigma)$.

Applying (6.9), the iteration number $C = 8\begin{pmatrix}3\\2\end{pmatrix} = 24$. Based on C, the length of output list can be determined as $l_2 = 3$ and parameter $t_2 = 1$ by using (6.11) and (6.12) respectively. As a result, the (1, 4)-weighted degree upper bound for the interpolated polynomial can be determined by (6.10) as $\max\{\deg_{1,4} Q\} = 13$. Therefore, the maximal pole basis (L_2) monomial that might exist in the interpolated polynomial is $\phi_{\max} = \phi_{12} = x^2y^3$. As the interpolation multiplicity $m = 2$, algorithm 6.1 is applied to determine the corresponding coefficients $\gamma_{0,p_i,\alpha} \sim \gamma_{12,p_i,\alpha}$ and $\gamma_{0,p_i,\alpha} \sim \gamma_{12,p_i,\alpha}$ $(\alpha < 2)$

are stored for the following interpolation process. Table 6.3 lists all the resulted corresponding coefficients $\gamma_{0,p_i,\alpha}$ ~ $\gamma_{12,p_i,\alpha}$ ($\alpha < 2$).

p_0, $\gamma_{a,p_0,\alpha}$	a α	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	1	0	0	0	0	0	0	0	0	0	0	0
p_1, $\gamma_{a,p_1,\alpha}$	a α	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	0	1	0	0	1	0	0	1	0	0	1	0
	1	0	1	0	0	1	0	0	1	0	0	1	0	0
p_2, $\gamma_{a,p_2,\alpha}$	a α	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	σ	1	σ	σ^2	σ	σ^2	1	σ^2	1	σ	1
	1	0	1	1	0	σ^2	0	1	σ^2	σ^2	0	σ	0	σ^2
p_3, $\gamma_{a,p_3,\alpha}$	a α	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	σ^2	1	σ^2	σ	σ^2	σ	1	σ	1	σ^2	1
	1	0	1	1	0	σ	0	1	σ	σ	0	σ^2	0	σ
p_4, $\gamma_{a,p_4,\alpha}$	a α	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	σ	σ	σ^2	σ^2	σ^2	1	1	1	σ	σ	σ	σ^2
	1	0	1	σ^2	0	σ^2	0	σ	σ^2	σ	0	σ	0	1
p_5, $\gamma_{a,p_5,\alpha}$	a α	0	1	2	3	4	5	6	7	8	9	10	11	12

	0	1	σ	σ^2	σ^2	1	σ	σ	σ^2	1	1	σ	σ^2	σ^2
	1	0	1	σ^2	0	σ	0	σ	σ	1	0	σ^2	0	σ^2
p_{6}, $\gamma_{a,p_6,\alpha}$	α \ a	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	σ^2	σ	σ	1	σ^2	σ^2	σ	1	1	σ^2	σ	σ
	1	0	1	σ	0	σ^2	0	σ^2	σ^2	1	0	σ	0	σ
p_{7}, $\gamma_{a,p_7,\alpha}$	α \ a	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	σ^2	σ^2	σ	σ	σ	1	1	1	σ^2	σ^2	σ^2	σ
	1	0	1	σ	0	σ	0	σ^2	σ	σ^2	0	σ^2	0	1

Table 6.3 Pre-determined corresponding coefficients for example 6.3

Following on, algorithm 6.2 is performed to find the interpolated polynomial $Q(x, y, z)$. At the beginning, a group of polynomials is initialised as:

$Q_0 = 1$, $Q_1 = y$, $Q_2 = z$, $Q_3 = yz$, $Q_4 = z^2$, $Q_5 = yz^2$, $Q_6 = z^3$, $Q_7 = yz^3$. Their leading orders are: $\text{lod}(Q_0) = 0$, $\text{lod}(Q_1) = 2$, $\text{lod}(Q_2) = 4$, $\text{lod}(Q_3) = 9$, $\text{lod}(Q_4) = 12$, $\text{lod}(Q_5) = 20$, $\text{lod}(Q_6) = 24$, $\text{lod}(Q_7) = 35$.

For interpolated unit $(p_0, r_0) = ((0, 0), 1)$,

For zero parameter $\alpha = 0$ and $\beta = 0$,

As $\text{lod}(Q_7) > C$, polynomial Q_7 is eliminated from the group.

Test the zero condition of the remaining polynomials in the group as:

$\Delta_0 = D_{00}^{(p_0,r_0)}(Q_0) = 1$, $\Delta_1 = D_{00}^{(p_0,r_0)}(Q_1) = 0$, $\Delta_2 = D_{00}^{(p_0,r_0)}(Q_2) = 1$, $\Delta_3 = D_{00}^{(p_0,r_0)}(Q_3) = 0$, $\Delta_4 = D_{00}^{(p_0,r_0)}(Q_4) = 1$, $\Delta_5 = D_{00}^{(p_0,r_0)}(Q_5) = 0$, $\Delta_6 = D_{00}^{(p_0,r_0)}(Q_6) = 1$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$j' = 0$ and $Q' = Q_0$

As $\Delta_1 = \Delta_3 = \Delta_5 = 0$

$Q_1 = Q_1 = y$, and lod(Q_1) = 2

$Q_3 = Q_3 = yz$, and lod(Q_3) = 9

$Q_5 = Q_5 = yz^2$, and lod(Q_5) = 20

Modify polynomials in the group with $\Delta_j \neq 0$ as:

$Q_0 = (x - 0)Q' = x$, and lod(Q_0) = 1

$Q_2 = \Delta_0 Q_2 - \Delta_2 Q' = 1 + z$, and lod($Q_2$) = 4

$Q_4 = \Delta_0 Q_4 - \Delta_4 Q' = 1 + z^2$, and lod($Q_4$) = 12

$Q_6 = \Delta_0 Q_6 - \Delta_6 Q' = 1 + z^3$, and lod($Q_6$) = 24

For zero parameter $\alpha = 0$ and $\beta = 1$,

As there is no polynomial in the group with leading order over C, no polynomial is eliminated in this iteration.

Test the zero condition of the remaining polynomials in the group as:

$\Delta_0 = D_{01}^{(p_0,r_0)}(Q_0) = 0$, $\Delta_1 = D_{01}^{(p_0,r_0)}(Q_1) = 0$, $\Delta_2 = D_{01}^{(p_0,r_0)}(Q_2) = 1$, $\Delta_3 = D_{01}^{(p_0,r_0)}(Q_3) = 0$, $\Delta_4 = D_{01}^{(p_0,r_0)}(Q_4) = 0$, $\Delta_5 = D_{01}^{(p_0,r_0)}(Q_5) = 0$, $\Delta_6 = D_{01}^{(p_0,r_0)}(Q_6) = 1$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$j' = 2$ and $Q' = Q_2$

As $\Delta_0 = \Delta_1 = \Delta_3 = \Delta_4 = \Delta_5 = 0$,

$Q_0 = Q_0 = x$, and lod(Q_0) = 1

$Q_1 = Q_1 = y$, and lod(Q_1) = 2

$Q_3 = Q_3 = yz$, and lod(Q_3) = 9

$Q_4 = Q_4 = 1 + z^2$, and lod(Q_4) = 12

$Q_5 = Q_5 = yz^2$, and lod(Q_5) = 20

Modify polynomials in the group with $\Delta_j \neq 0$ as:

$Q_2 = (x - 0)Q' = x + xz$, and lod(Q_2) = 7

$Q_6 = \Delta_2 Q_6 - \Delta_6 Q' = z + z^3$, and lod($Q_6$) = 24

For zero parameter $\alpha = 1$ and $\beta = 0$,

As there is no polynomial in the group with leading order over C, no polynomial is eliminated in this iteration.

Test the zero condition of the remaining polynomials in the group as:

$\Delta_0 = D_{10}^{(p_0,r_0)}(Q_0) = 1$, $\Delta_1 = D_{10}^{(p_0,r_0)}(Q_1) = 0$, $\Delta_2 = D_{10}^{(p_0,r_0)}(Q_2) = 0$, $\Delta_3 = D_{10}^{(p_0,r_0)}(Q_3) = 0$, $\Delta_4 = D_{10}^{(p_0,r_0)}(Q_4) = 0$, $\Delta_5 = D_{10}^{(p_0,r_0)}(Q_5) = 0$, $\Delta_6 = D_{10}^{(p_0,r_0)}(Q_6) = 0$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$j' = 0$ and $Q' = Q_0$

As $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = \Delta_6 = 0$,

$Q_1 = Q_1 = y$, and $\text{lod}(Q_1) = 2$

$Q_2 = Q_2 = x + xz$, and $\text{lod}(Q_2) = 7$

$Q_3 = Q_3 = yz$, and $\text{lod}(Q_3) = 9$

$Q_4 = Q_4 = 1 + z^2$, and $\text{lod}(Q_4) = 12$

$Q_5 = Q_5 = yz^2$, and $\text{lod}(Q_5) = 20$

$Q_6 = Q_6 = z + z^3$, and $\text{lod}(Q_6) = 24$

Modify polynomials in the group with $\Delta_j \neq 0$ as:

$Q_0 = (x - 0)Q' = x^2$, and $\text{lod}(Q_0) = 3$

Following the same process, interpolation runs through the rest of the interpolated units $(p_1, r_1) \sim (p_7, r_7)$ and with respect to all zero parameters $(\alpha, \beta) = (0, 0), (0, 1)$, and $(1, 0)$. After C iterations, the chosen interpolated polynomial is: $Q(x, y, z) = \sigma^2 + \sigma x + y + \sigma x^2 + y^2 + \sigma^2x^2y + \sigma^2xy^2 + \sigma^2y^3 + \sigma^2x^2y^2 + z(x + xy + xy^2) + z^2(\sigma^2 + \sigma x + \sigma^2 y + \sigma x^2)$, and $\text{lod}(Q(x, y, z)) = 23$. Polynomial $Q(x, y, z)$ has a zero of multiplicity at least 2 over the 8 interpolated units.

6.6 General Factorisation

Based on the interpolated polynomial, factorisation is to find the polynomial's *z* roots in order to determine the output list. Building upon the work of [10] and [12], this section presents a generalised factorisation algorithm, or so called the recursive coefficient search algorithm which can be applied to both Reed-Solomon codes and algebraic-geometric codes. This section's work is based on the author's paper [67]. In general, the algorithm is described with application to algebraic-geometric codes. Therefore, it has to be stated that when applying this algorithm to Reed-Solomon codes, the rational functions in an algebraic-geometric code's pole basis are simplified to univariate monomials in Reed-Solomon code's pole basis. As a consequence polynomials in $F_q[x, y]$ are simplified to univariate polynomials with variable x.

Based on section 6.3, those polynomials $h \in F_q^{w_z}[x, y]$ will be in the output list if $Q(x, y, h) = 0$. The outcome of the factorisation can be written as:

$$\begin{cases} h_1 = h_{1,0}\phi_0 + \cdots + h_{1,k-1}\phi_{k-1} \\ \qquad \vdots \\ h_l = h_{l,0}\phi_0 + \cdots + h_{l,k-1}\phi_{k-1} \end{cases} \tag{6.36}$$

with $l \le l_m$. Rational functions $\phi_0, \ldots, \phi_{k-1}$ are predetermined by the decoder, therefore, to find the list of polynomials is equivalent to find their coefficients $h_{1,0}, \ldots, h_{1,k-1}, \ldots, h_{l,0}, \ldots, h_{l,k-1}$ respectively. Substituting h into the interpolated polynomial $Q = \sum_{a,b \in \mathrm{N}} Q_{ab}\phi_a z^b$, we have:

$$Q(x, y, h) = \sum_{a,b \in \mathrm{N}} Q_{ab}\phi_a h^b = \sum_{a,b \in \mathrm{N}} Q_{ab}\phi_a (h_0\phi_0 + \cdots + h_{k-1}\phi_{k-1})^b \tag{6.37}$$

It is important to notice that:

$$(\phi_i \phi_j) \bmod \chi = \sum_{v \in \mathrm{N}} \phi_v \tag{6.38}$$

where χ is the algebraic curve (e. g. Hermitian curve H_w) and ϕ_i, ϕ_j, and ϕ_v are rational functions in pole basis which is associated with the curve χ (e. g. pole basis L_w associated with curve H_w). Therefore (6.37) can be re-written as a polynomial in $F_q[x, y]$:

$$Q(x, y, h) = \sum_{a \in \mathrm{N}} Q_a \phi_a \tag{6.39}$$

where coefficients Q_a are equations with unknowns $h_0, \ldots, h_{k-1}$. If $T = |\{Q_a \mid Q_a \neq 0\}|$, the rational functions ϕ_a with $Q_a \neq 0$ can be arranged as $\phi_{a_1} < \phi_{a_2} < \cdots < \phi_{a_T}$ and (6.39) can again be written as:

$$Q(x, y, h) = Q_{a_1}\phi_{a_1} + Q_{a_2}\phi_{a_2} + \cdots + Q_{a_T}\phi_{a_T} \tag{6.40}$$

Again, coefficients $Q_{a_1}, Q_{a_2}, \ldots,$ and Q_{a_T} are equations of unknowns $h_0, \ldots, h_{k-1}$. To have $Q(x, y, h) = 0$, we need $Q_{a_1} = Q_{a_2} = \cdots = Q_{a_T} = 0$. Therefore, $h_0, \ldots, h_{k-1}$ can be determined by solving the following simultaneous set of equations as:

$$\begin{cases} Q_{a_1}(h_0, \ldots, h_{k-1}) = 0 \\ Q_{a_2}(h_0, \ldots, h_{k-1}) = 0 \\ \qquad \vdots \\ Q_{a_T}(h_0, \ldots, h_{k-1}) = 0 \end{cases} \tag{6.41}$$

In order to solve equation set (6.41), a recursive coefficient search algorithm is applied to determine $h_0, \ldots, h_{k-1}$ [11, 12]. Following on, here is to propose a more general and efficient factorisation algorithm. Let us denote the following polynomials with respect to recursive index s ($0 \leq s \leq k - 1$):

$$h^{(s)}(x, y) = h_0\phi_0 + \cdots + h_{k-1-s}\phi_{k-1-s} \tag{6.42}$$

which is a candidate polynomial with coefficients $h_0, .., h_{k-1-s}$ undetermined. Update $Q(x, y, z)$ recursively as:

$$Q^{(s+1)}(x, y, z) = Q^{(s)}(x, y, z + h_{k-1-s}\phi_{k-1-s}) \tag{6.43}$$

with $Q^{(0)}(x, y, z) = Q(x, y, z)$ which is the interpolated polynomial (6.8). Substitute $h_{k-1-s}\phi_{k-1-s}$ into $Q^{(s)}(x, y, z)$, we have:

$$\widetilde{Q}^{(s)}(x, y) = Q^{(s)}(x, y, h_{k-1-s}\phi_{k-1-s}) \tag{6.44}$$

With $\widetilde{Q}^{(s)}$ mod χ, it can be transferred into a polynomial in $F_q[x, y]$ with coefficients expressed as $\sum_{i \in \mathrm{N}} \varpi_i h_{k-1-s}^i$ where $\varpi_i \in \mathrm{GF}(q)$. Denote $\widetilde{Q}^{(s)}$'s leading monomial with its leading coefficient as:

$$\phi_L^{(s)} = \mathrm{LM}(\widetilde{Q}^{(s)}) \tag{6.45}$$

$$C_L^{(s)}(h_{k-1-s}) = \mathrm{LC}(\widetilde{Q}^{(s)}) \tag{6.46}$$

Based on (6.42), it can be seen that $\mathrm{LM}(h^{(s)}) = \phi_{k\text{-}1\text{-}s}$ and $\mathrm{LC}(h^{(s)}) = h_{k\text{-}1\text{-}s}$. Therefore, for any recursive polynomial $Q^{(s)}(x, y, z)$, we have:

$$\mathrm{LM}(Q^{(s)}(x, y, h^{(s)})) = \mathrm{LM}(Q^{(s)}(x, y, h_{k\text{-}1\text{-}s}\phi_{k\text{-}1\text{-}s})) = \mathrm{LM}(\widetilde{Q}^{(s)}) = \phi_L^{(s)} \tag{6.47}$$

$$\mathrm{LC}(Q^{(s)}(x, y, h^{(s)})) = \mathrm{LC}(Q^{(s)}(x, y, h_{k\text{-}1\text{-}s}\phi_{k\text{-}1\text{-}s})) = \mathrm{LC}(\widetilde{Q}^{(s)}) = C_L^{(s)}(h_{k-1-s}) \tag{6.48}$$

As all the candidate outputs should satisfy $Q(x, y, h) = 0$ and from the above definitions it can be seen that $h = h^{(0)}$ and $Q^{(0)}(x, y, z) = Q(x, y, z)$, therefore $Q(x, y, h) = 0$ is equivalent to $Q^{(0)}(x, y, h^{(0)}) = 0$. Based on (6.47) and (6.48), in order to have $Q^{(0)}(x, y, h^{(0)}) = 0$, we need to find out its leading monomial $\phi_L^{(0)}$ with leading coefficient $C_L^{(0)}(h_{k-1})$ and determine values of $h_{k\text{-}1}$ that satisfy $C_L^{(0)}(h_{k-1}) = 0$. As a result, the leading monomial of $Q^{(0)}(x, y, h^{(0)})$ has been eliminated. Based on each value of $h_{k\text{-}1}$ and performing the polynomial update (6.43), $Q^{(1)}(x, y, z)$ is generated, in which $\phi_L^{(0)}$ has been eliminated. Now, $Q(x, y, h) = 0$ is equivalent to $Q^{(1)}(x, y, h^{(1)})=0$. Again, to have $Q^{(1)}(x, y, h^{(1)}) = 0$, we need $C_L^{(1)}(h_{k-2}) = 0$. Therefore, $h_{k\text{-}2}$ can be determined by solving $C_L^{(1)}(h_{k-2}) = 0$. Based on each value of $h_{k\text{-}2}$, we can trace further to find the rest of the coefficients. In general, after coefficients $h_{k\text{-}1\text{-}s}$ $(0 \leq s < k - 1)$ have been determined from solving $C_L^{(s)}(h_{k-1-s}) = 0$, based on each value of them, perform the polynomial update (6.43) to generate $Q^{(s+1)}(x, y, z)$. From $Q^{(s+1)}(x, y, z)$, $\widetilde{Q}^{(s+1)}$ can be calculated and $h_{k\text{-}1\text{-}(s+1)}$ can be determined by solving $C_L^{(s+1)}(h_{k-1-(s+1)}) = 0$. This process can be illustrated in Fig. 6.2:

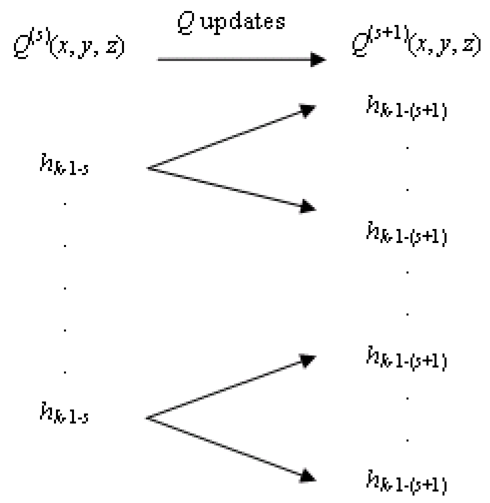


Figure 6.2 Recursive coefficient search

From Fig. 6.2 it can be seen that there might be an exponential number of routes to find coefficients h_{k-1}, …, h_0. However, not every route will be able to reach h_0 as during the recursive process there maybe no solution for $C_L^{(s)}(h_{k-1-s}) = 0$. If h_0 is produced and $Q^{(k-1)}(x, y, h_0\phi_0) = 0$, this route can be traced to find the rest of the coefficients h_1, …, h_{k-1} to construct polynomial h which will satisfy $Q(x, y, h) = 0$. The correctness of this judgement will be proven later.

Based on the above description, here summarises the generalised factorisation algorithm as:

Algorithm 6.3: Recursive Coefficient Search

Initialisation: $Q^{(0)}(x, y, z) = Q(x, y, z)$. The recursive index $s = 0$ and output candidate index $l = 1$

Perform: Recursive coefficient search (s) ($RCS(s)$)

Recursive coefficient search (*RCS*):

Input parameter: s ($0 \le s \le k - 1$)

(i): Perform (6.44) to calculate $\widetilde{Q}^{(s)}(x, y)$

(ii): Find out $\phi_L^{(s)}$ with its coefficient $C_L^{(s)}(h_{k-1-s})$

```
(iii): Determine $h_{k-1-s}$ by solving $C_L^{(s)}(h_{k-1-s}) = 0$
(iv): For each value of $h_{k-1-s}$, do
{
(v):      $h_{l, k-1-s} = h_{k-1-s}$
(vi):     If $s = k - 1$, calculate $Q^{(k-1)}(x, y, h_0\phi_0)$ and go to (vii). Else, go to (viii)
(vii):    If $Q^{(k-1)}(x, y, h_0\phi_0) = 0$, trace this route to find coefficients $h_{l, k-1}$, $h_{l, k-2}$, …, and
          $h_{l, 0}$ to construct the candidate polynomial $h_l$ and $l = l + 1$. Else, stop this route
(viii):   Perform polynomial update (6.43) to generate $Q^{(s+1)}(x, y, z)$
(ix):     Perform $RCS(s + 1)$
}
```

This recursive coefficient search algorithm has the priority to search deeper coefficients. This means if a number of h_{k-1-s} have been determined, the algorithm will be based on one of them to determine the rest of the coefficients until all the possible routes started from this h_{k-1-s} have been traced. After this, it will be based on the other value of h_{k-1-s} and trace deeper again. This algorithm will terminate after all the possible routes started from h_{k-1} have been traced. To prove the correctness of this algorithm, it needs to justify the polynomial h_l produced in (vii) satisfies $Q(x, y, h_l) = 0$.

Proof: As $Q^{(k-1)}(x, y, h_0\phi_0) = 0$ and $h^{(k-1)}(x, y) = h_0\phi_0$, we have $Q^{(k-1)}(x, y, h^{(k-1)}) = 0$. Assuming h_1 is the previously found coefficient, then $Q^{(k-1)}(x, y, z)$ is generated by (6.43) based on it as: $Q^{(k-1)}(x, y, z) = Q^{(k-2)}(x, y, z + h_1\phi_1)$. From $Q^{(k-1)}(x, y, h^{(k-1)}) = 0$, we have $Q^{(k-2)}(x, y, h^{(k-1)} + h_1\phi_1) = 0$. Based on (6.42), it can be seen that $h^{(k-2)} = h_0\phi_0 + h_1\phi_1 = h^{(k-1)} + h_1\phi_1$. Therefore, $Q^{(k-2)}(x, y, h^{(k-2)}) = 0$. Based on the same deduction progress, it can be deduced further to have $Q^{(k-3)}(x, y, h^{(k-3)}) = 0$, …, and $Q^{(0)}(x, y, h^{(0)}) = 0$. As $Q^{(0)}(x, y, z) = Q(z)$ and $h^{(0)} = h_0\phi_0 + h_1\phi_1 + \cdots + h_{k-1}\phi_{k-1}$ whose coefficients have been traced as the coefficients of the output candidate h_l, it can be concluded that $Q(x, y, h_l) = 0$.

Here gives two worked examples to illustrate the generalised factorisation algorithm's application to a Hermitian code and a Reed-Solomon code respectively.

Example 6.4 List decode a (8, 4, 4) Hermitian code defined in GF(4). Given the interpolated polynomial is: $Q(x, y, \mathrm{z}) = \sigma^2 y + \sigma^2 x^2 + \sigma xy + \sigma^2 y^2 + \sigma^2 x^2 y + x^2 y^2 + xy^3 + (\sigma x + \sigma xy + \sigma xy^2)z + (x + x^2)z^2$. Apply algorithm 6.3 to find out its z roots.

Initialisation: $Q^{(0)}(x, y, z) = Q(x, y, z)$, $s = 0$ and $l = 1$

$RCS(0)$:

$\widetilde{Q}^{(0)}(x, y) = Q^{(0)}(x, y, h_3 x^2) = (\sigma^2 + \sigma h_3)y + \sigma^2 x^2 + \sigma xy + (\sigma^2 + h_3^2)y^2 + (\sigma^2 + h_3^2)x^2 y + (1 + h_3^2)x^2 y^2 + xy^3 + (\sigma h_3 + h_3^2)y^4$, with $\phi_L^{(0)} = y^4$ and $C_L^{(0)}(h_3) = \sigma h_3 + h_3^2$. Solving $C_L^{(0)}(h_3) = 0$, we have $h_3 = 0$ or $h_3 = \sigma$.

For $h_3 = 0$, $h_{1,3} = h_3 = 0$. As $s = 0 < 3$, update $Q^{(1)}(x, y, z) = Q^{(0)}(x, y, z + 0x^2) = Q(x, y, z)$, and perform $RCS(1)$…

Based on the same progress, here summarises the outcome from $RCS(1)$, $RCS(2)$ and $RCS(3)$ in Table 6.4 as:

$RCS(s)$	$\phi_L^{(s)}$	$C_L^{(s)}(h_{k-1-s})$	$h_{1,k-1-s} = h_{k-1-s}$
$RCS(1)$	xy^3	$1 + \sigma h_2$	σ^2
$RCS(2)$	x^2y^2	$\sigma^2 + \sigma h_1$	σ
$RCS(3)$	xy^2	σh_0	0

Table 6.4 Recursive coefficient search from $h_3 = 0$

After $RCS(2)$, we have $Q^{(3)}(x, y, z) = (\sigma x + \sigma xy + \sigma xy^2)z + (x + x^2)z^2$. In $RCS(3)$, by solving $C_L^{(3)}(h_0) = 0$, we have $h_0 = 0$. Therefore, $h_{1,0} = h_0 = 0$. As $s = 3$ and $Q^{(3)}(x, y, h_0\phi_0) = Q^{(3)}(x, y, 0\cdot 1) = 0$, this route can be traced to construct candidate polynomial $h_1 = \sigma x + \sigma^2 y$, and update the candidate index $l = l + 1 = 2$.

Going back to the closest division point (when $s = 0$), we have:

For $h_3 = \sigma$, $h_{2,3} = h_3 = \sigma$. As $s = 0 < 3$, update $Q^{(1)}(x, y, z) = Q^{(0)}(x, y, z + \sigma x^2) = \sigma^2x^2 + \sigma xy + \sigma x^2y^2 + xy^3 + (\sigma x + \sigma xy + \sigma xy^2)z + (x + x^2)z^2$, and perform $RCS(1)$…

Again, here summarises the outcome of $RCS(1)$, $RCS(2)$ and $RCS(3)$ in Table 6.5 as:

$RCS(s)$	$\phi_L^{(s)}$	$C_L^{(s)}(h_{k-1-s})$	$h_{2,k-1-s} = h_{k-1-s}$
$RCS(1)$	xy^3	$1 + \sigma h_2$	σ^2
$RCS(2)$	x^2y^2	σh_1	0
$RCS(3)$	xy^2	$\sigma^2 + \sigma h_0$	σ

Table 6.5 Recursive coefficient search from $h_3 = \sigma$

After $RCS(2)$, we have $Q^{(3)}(x, y, z) = \sigma^2x^2 + \sigma^2xy + \sigma^2xy^2 + (\sigma x + \sigma xy + \sigma xy^2)z + (x + x^2)z^2$. In $RCS(3)$, by solving $C_L^{(3)}(h_0) = 0$, we have $h_{2,0} = h_0 = \sigma$. As $s = 3$ and $Q^{(3)}(x, y, h_0\phi_0) = Q^{(3)}(x, y, \sigma\cdot 1) = 0$, this route can be traced to construct the candidate polynomial $h_2 = \sigma + \sigma^2y + \sigma x^2$. As all the possible routes from h_0 have been traced, the factorisation process terminates and outputs: $h_1 = \sigma x + \sigma^2y$, $h_2 = \sigma + \sigma^2y + \sigma x^2$.

Example 6.5 List decoding of a (7, 2, 6) Reed-Solomon code defined in GF(8). σ is a primitive element in GF(8) satisfying $\sigma^3 + \sigma + 1 = 0$. Addition and multiplication table of GF(8) is given in Appendix B. Given the interpolated polynomial as: $Q(x, z) = \sigma x + \sigma^6x^2 + (\sigma^3 + \sigma^3x)z + \sigma^2z^2$. Apply algorithm 6.3 to determine its z roots.

Initialisation: $Q^{(0)}(x, z) = Q(x, z)$, $s = 0$ and $l = 1$

$RCS(0)$:

$\tilde{Q}^{(0)}(x, z) = Q^{(0)}(x, h_1x) = (\sigma + \sigma^3h_1)x + (\sigma^6 + \sigma^3h_1 + \sigma^2h_1^2)x^2$, with $\phi_L^{(0)} = x^2$ and $C_L^{(0)}(h_1) = \sigma^6 + \sigma^3h_1 + \sigma^2h_1^2$. Solving $C_L^{(0)}(h_1) = 0$, we have $h_1 = \sigma^5$ or $h_1 = \sigma^6$.

For $h_1 = \sigma^5$, $h_{1,1} = h_1 = \sigma^5$. As $s = 0 < 1$, update $Q^{(1)}(x, z) = Q^{(0)}(x, z + \sigma^5x) = (\sigma^3 + \sigma^3x)z + \sigma^2z^2$, and perform $RCS(1)$

In *RCS*(1), following the same progress, we have $\phi_L^{(1)} = x$ and $C_L^{(1)}(h_0) = \sigma^3 h_0$. Solving $C_L^{(1)}(h_0) = 0$, we have $h_0 = 0$.

For $h_0 = 0$, $h_{1,0} = h_0 = 0$. As $s = 1$ and $Q^{(1)}(x, h_0\phi_0) = Q^{(1)}(x, 0\cdot 1) = 0$, this route can be traced to construct candidate polynomial $h_1 = \sigma^5 x$. Update the output candidate index as $l = l + 1 = 2$

Going back to the closest division point (when $s = 0$), we have:

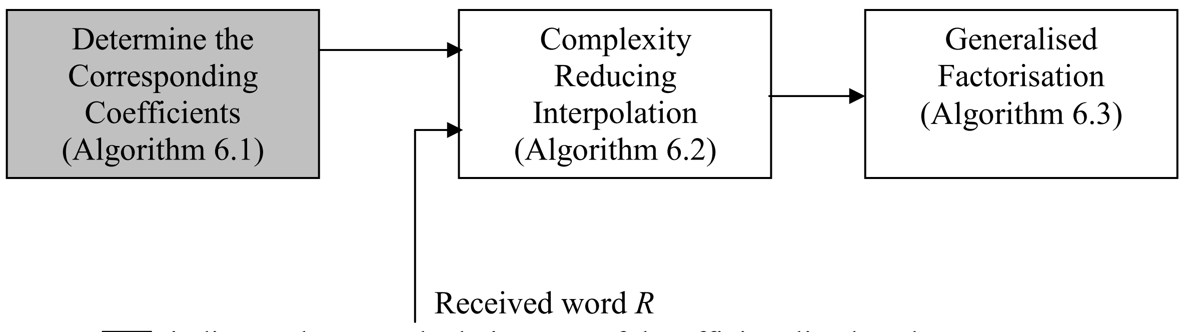
For $h_1 = \sigma^6$, $h_{2,1} = h_1 = \sigma^6$. As $s = 0 < 1$, update $Q^{(1)}(x, z) = Q^{(0)}(x, z + \sigma^6 x) = \sigma^4 x + (\sigma^3 + \sigma^3 x)z + \sigma^2 z^2$, and perform *RCS*(1)

In *RCS*(1), we have $\phi_L^{(1)} = x$ and $C_L^{(1)}(h_0) = \sigma^4 + \sigma^3 h_0$. Solving $C_L^{(1)}(h_0) = 0$, we have $h_0 = \sigma$.

For $h_0 = \sigma$, $h_{2,0} = h_0 = \sigma$. As $s = 1$ and $Q^{(1)}(x, h_0\phi_0) = Q^{(1)}(x, \sigma\cdot 1) = 0$, this route can be traced to construct candidate polynomial $h_2 = \sigma + \sigma^6 x$. As all the possible routes from h_0 have been traced, the factorisation process terminates and outputs: $h_1 = \sigma^5 x$, $h_2 = \sigma + \sigma^6 x$.

6.7 Simulation Results Discussion

Employing the above efficiency improved interpolation algorithm and generalised factorisation algorithm, list decoding of longer Hermitian codes is feasible and the author has developed a software platform using the C programming language to evaluate the performance. The evaluating list decoder structure is shown by Fig. 6.3. Simulations are run over AWGN and Rayleigh fading channels using QPSK modulation. The Rayleigh fading channel is frequency non-selective with Doppler frequency 126.67 Hz and date rate of 30 kb/s. The fading profile is generated by Jakes' method [64]. The fading coefficients have mean value 1.56 and variance 0.60. Over the fading channel, a block interleaver with size $100 \times n$ is used, where n is the length of the code. Simulation results are analysed with regard to the following two aspects: performance comparison with the conventional unique decoding algorithm and performance comparison with the Reed-Solomon codes using list decoding.


Determine the Corresponding Coefficients (Algorithm 6.1)
Complexity Reducing Interpolation (Algorithm 6.2)
Generalised Factorisation (Algorithm 6.3)
Received word R

Note: ▨ indicates the pre-calculation step of the efficient list decoder.

Figure 6.3 Efficiency improved list decoding structure for Hermitian codes

6.7.1 Comparison with Unique Decoding Algorithm

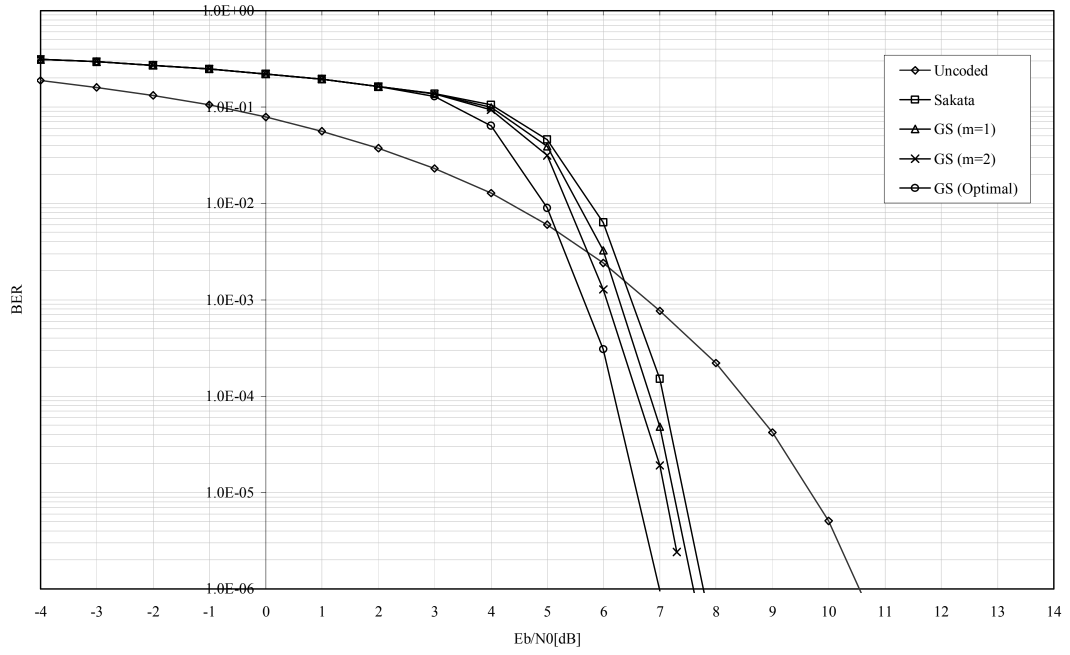
Figs. 6.4, 6.5 and 6.6 present the performance of Hermitian codes (64, 19, 40), (64, 29, 30) and (64, 39, 20) respectively, while Figs. 6.7 and 6.8 present the performance of Hermitian codes (512, 153, 332) and (512, 289, 196) respectively. These simulation results were first published in the author's papers [65, 66]. Their performances are evaluated by measuring their coding gains (dB) over the unique decoding algorithm [5, 6, 25] at a bit error rate (BER) of 10^{-5}.

According to the interpolation description in section 6.5, for interpolation with multiplicity m, there are $w(l_m + 1)$ polynomials being initialised taking part in C (6.9) iterations. Even though some of them will be eliminated during the iteration by scheme (6.29), a high value of m will still lead to infeasibility for implementation. Therefore, some feasible results of the GS algorithm (m = 1, 2) have been achieved. Table 6.6 presents the simulation parameters for these 5 Hermitian codes. Also, details of more list decoding parameters of these 5 Hermitian codes are included in Appendix D. Before interpolation, algorithm 6.1 is applied to determine the corresponding coefficients $\gamma_{0,p_i,\alpha} \sim \gamma_{\max,p_i,\alpha}$ and only $\gamma_{0,p_i,\alpha} \sim \gamma_{\max,p_i,\alpha}$ (α = 0, 1) are stored in order to minimise the memory requirement. From Table 6.6, it can be observed that achieving the optimal result of the GS algorithm remains prohibitive for implementation. But if assuming the GS algorithm is able to correct τ_{GS} (3.32) errors and the transmitted code word $\bar{c}$ is known by the decoder, the theoretical optimal performance of the GS algorithm can also be evaluated without employing the interpolation and factorisation processes. This is achieved by measuring the Hamming distance between the received word *R* and the transmitted code word $\bar{c}$. If it is not greater than τ_{GS}, decoding is claimed to be successful. Otherwise, decoding is a failure. Figs. 6.4 to 6.8 show that the GS algorithm approaches its optimal result with increasing interpolation multiplicity *m*. Among the performance evaluations, it is worth pointing out Fig. 6.6 which shows that GS decoding of Hermitian code (64, 39, 20) with multiplicity *m* = 2 is close to the theoretical optimal result.

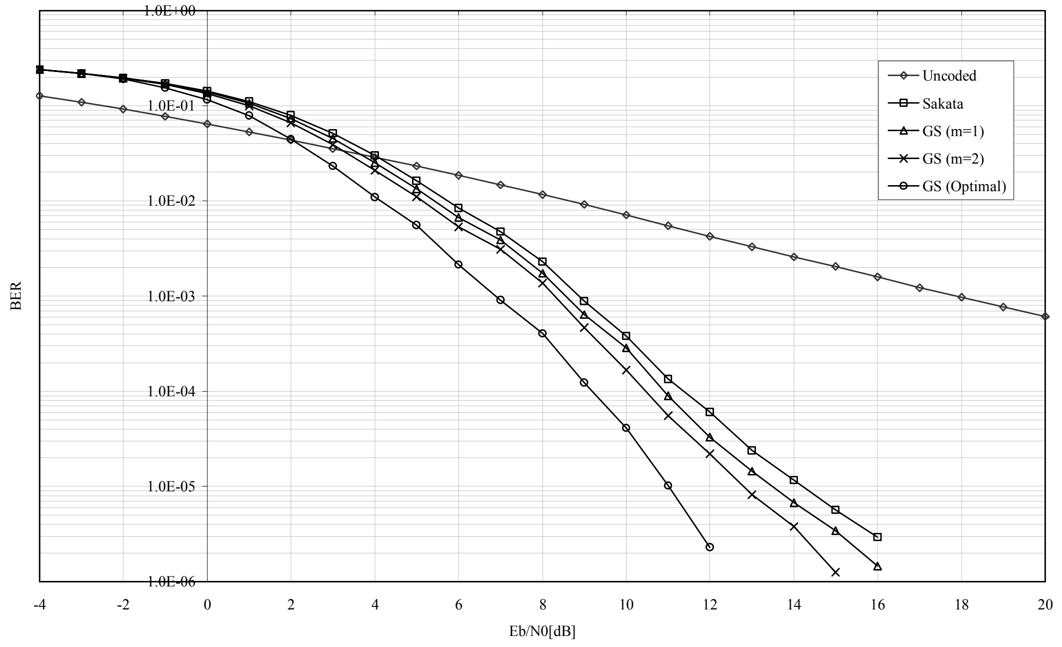
Hermitian codes	Interpolation multiplicity	C	l_m	$w(l_m + 1)$	$\max\{\deg_{1,w_z} Q\}$	$\phi_{\max}$
(64, 19, 40)	$m = 1$	64	2	12	50	$\phi_{44} = y^{10}$
	$m = 2$	192	3	16	90	$\phi_{84} = y^{18}$
	optimal (m = 17)	9792	28	116	—	—
(64, 29, 30)	$m = 1$	64	1	8	55	$\phi_{49} = y^{55}$
	$m = 2$	192	3	16	104	$\phi_{98} = xy^{20}$
	optimal (m = 35)	40320	48	196	—	—
(64, 39, 20)	$m = 1$	64	1	8	60	$\phi_{54} = y^{12}$
	$m = 2$	192	2	12	114	$\phi_{108} = xy^{22}$
	optimal (m = 11)	4224	13	56	—	—
(512, 153, 332)	$m = 1$	512	2	24	373	$\phi_{345} = x^5y^{37}$
	$m = 2$	1536	3	32	682	$\phi_{654} = x^2y^{74}$
	optimal (m = 213)	11668992	359	2880	—	—
(512, 289, 196)	$m = 1$	512	1	16	442	$\phi_{414} = x^8y^{42}$
	$m = 2$	1536	2	24	856	$\phi_{828} = x^8y^{88}$
	optimal (m = 93)	2237952	118	952	—	—

Note: C represents the number of iterations; l_m represents the maximal length of the output list from factorisation;, $w(l_m + 1)$ represents the number of polynomials being initialised at the beginning of the interpolation process. $\max\{\deg_{1,w_z} Q\}$ represents the interpolated polynomial's $(1, w_z)$-weighted degree upper bound, $\phi_{\max}$ represents the maximal pole basis monomial that might exist in the interpolated polynomial Q.

Table 6.6 List decoding parameters

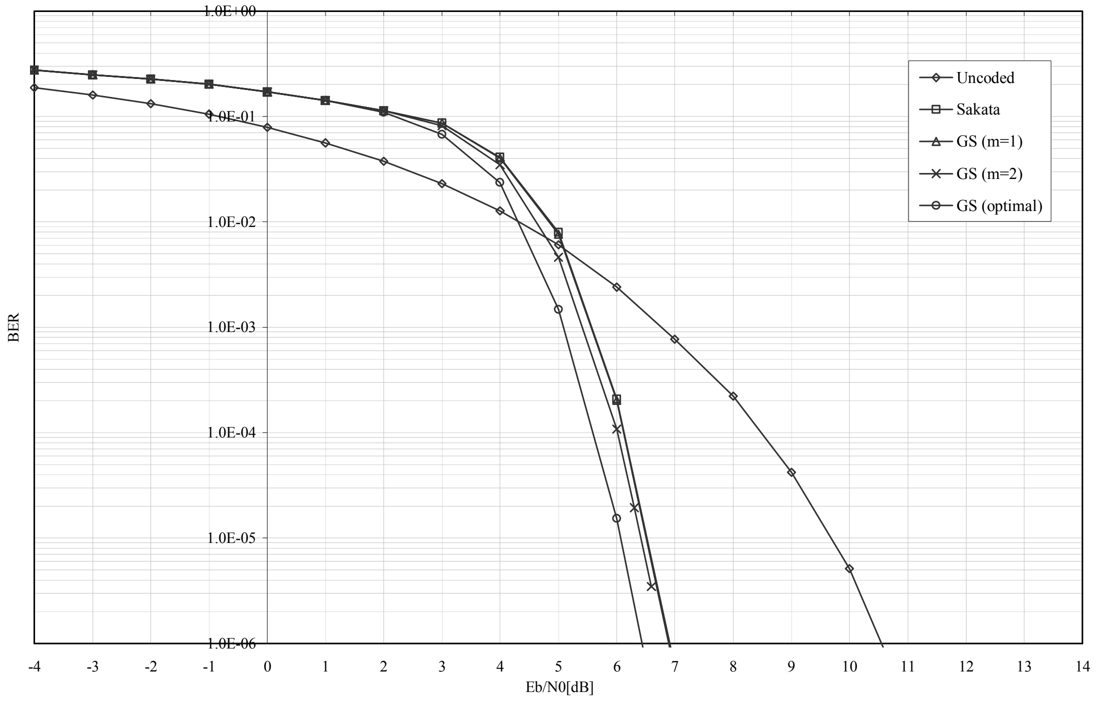


(a) over AWGN channel

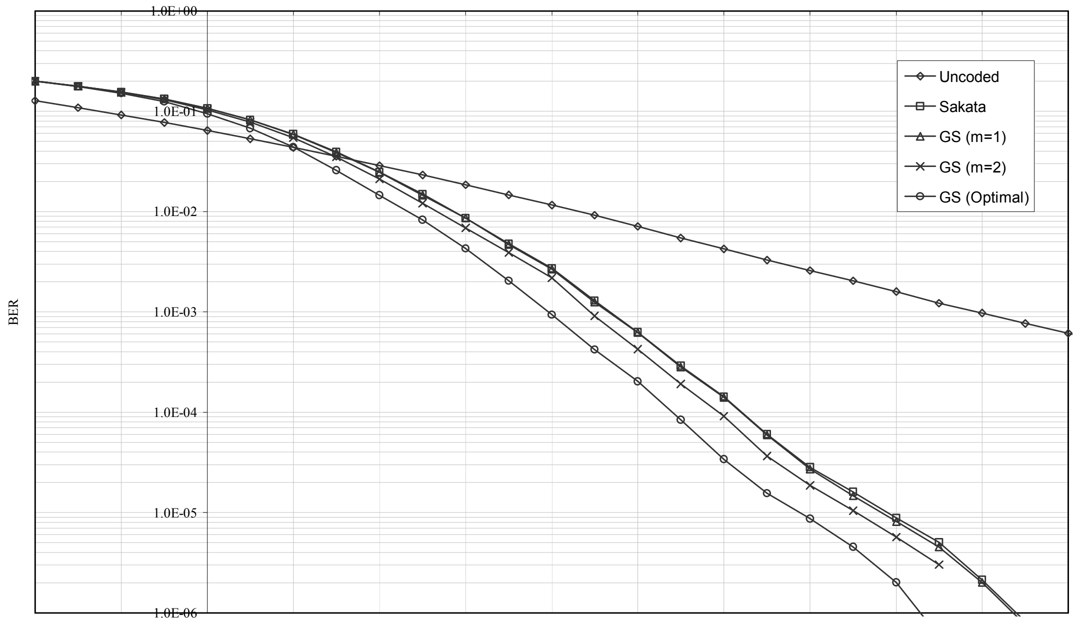


(b) over Rayleigh fading channel

Figure 6.4 Hard-decision list decoding performance of Hermitian code (64, 19, 20)

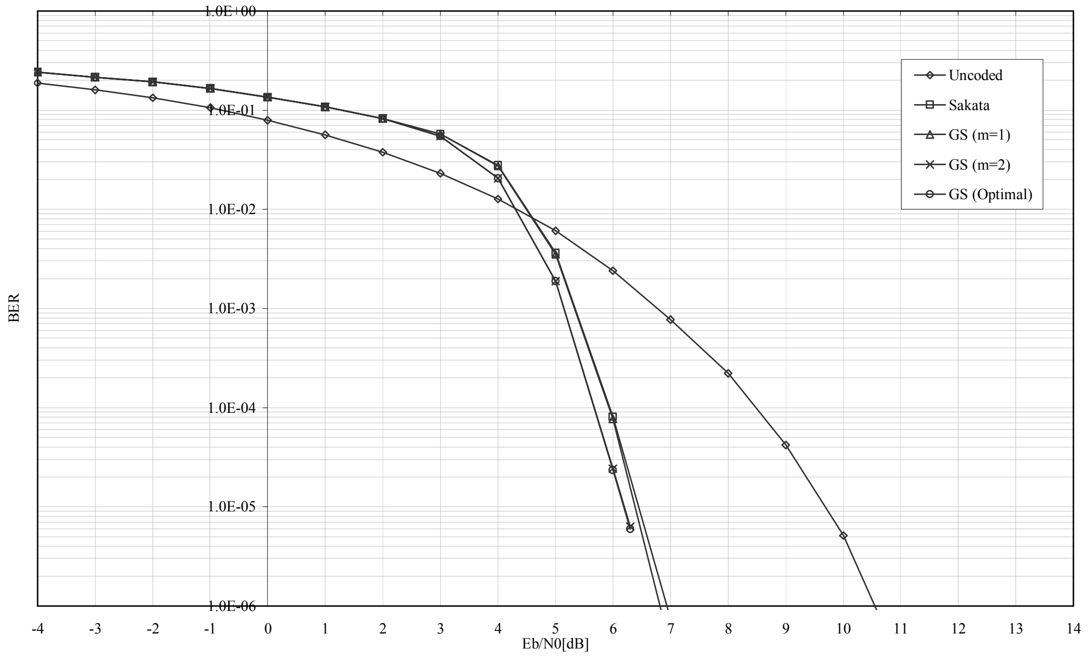


(a) over AWGN channel

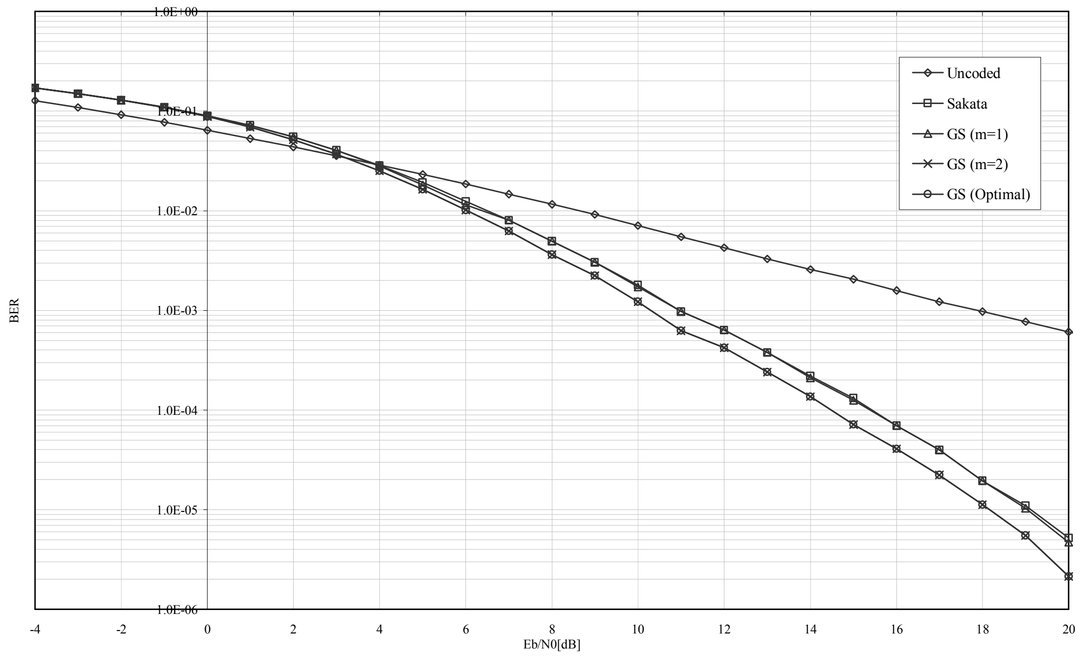


(b) over Rayleigh fading channel

Figure 6.5 Hard-decision list decoding performance of Hermitian code (64, 29, 30)

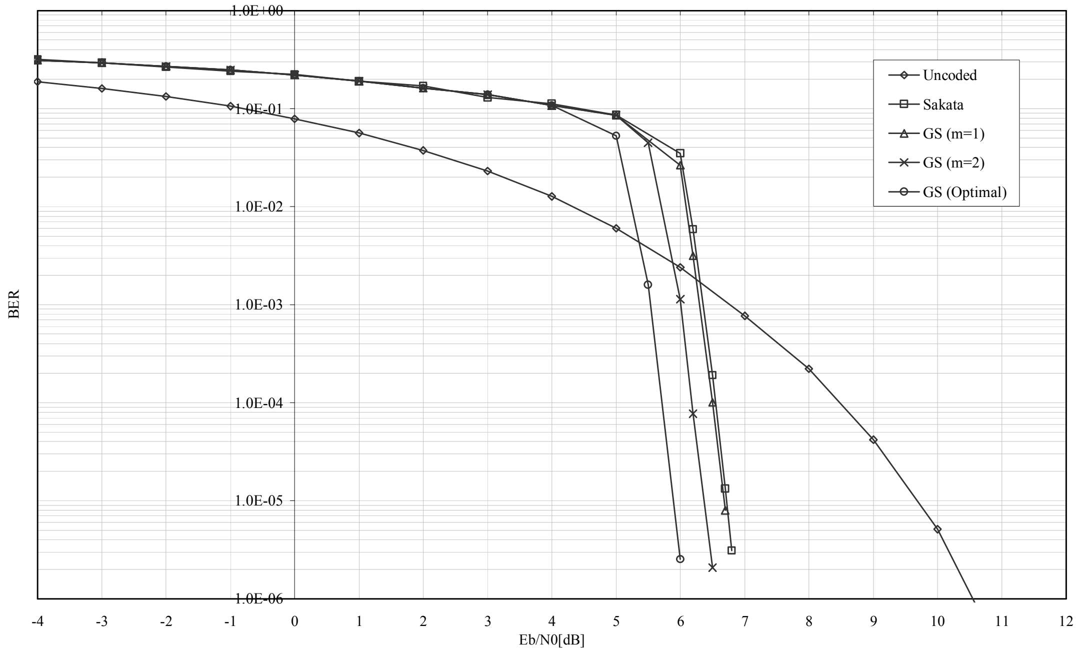


(a) over AWGN channel

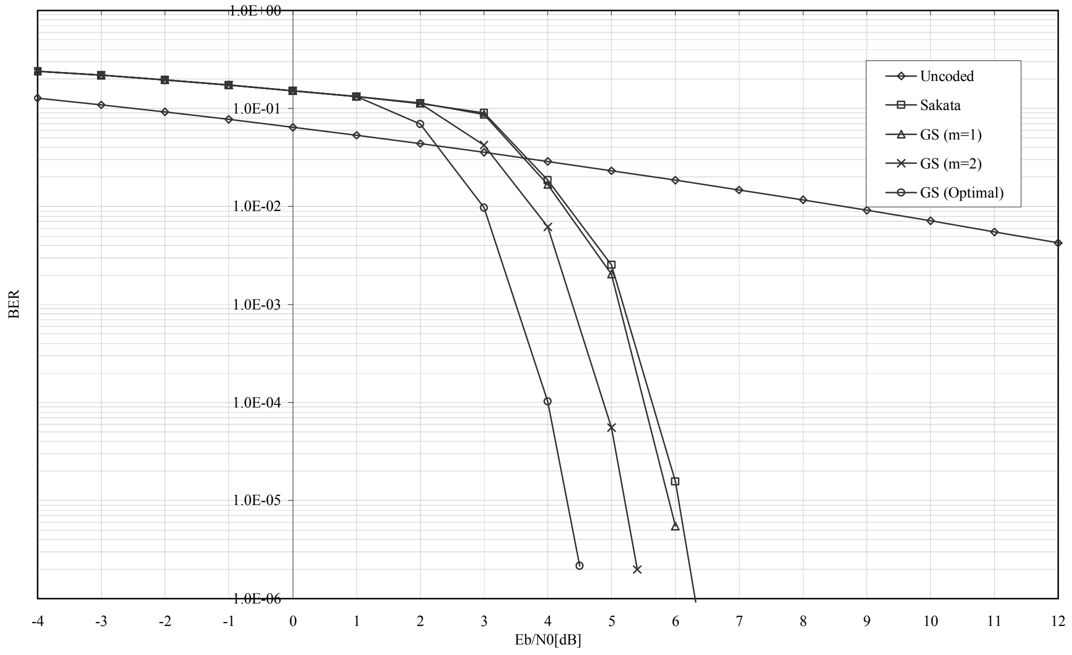


(b) over Rayleigh fading channel

Figure 6.6 Hard-decision list decoding performance of Hermitian code (64, 39, 20)

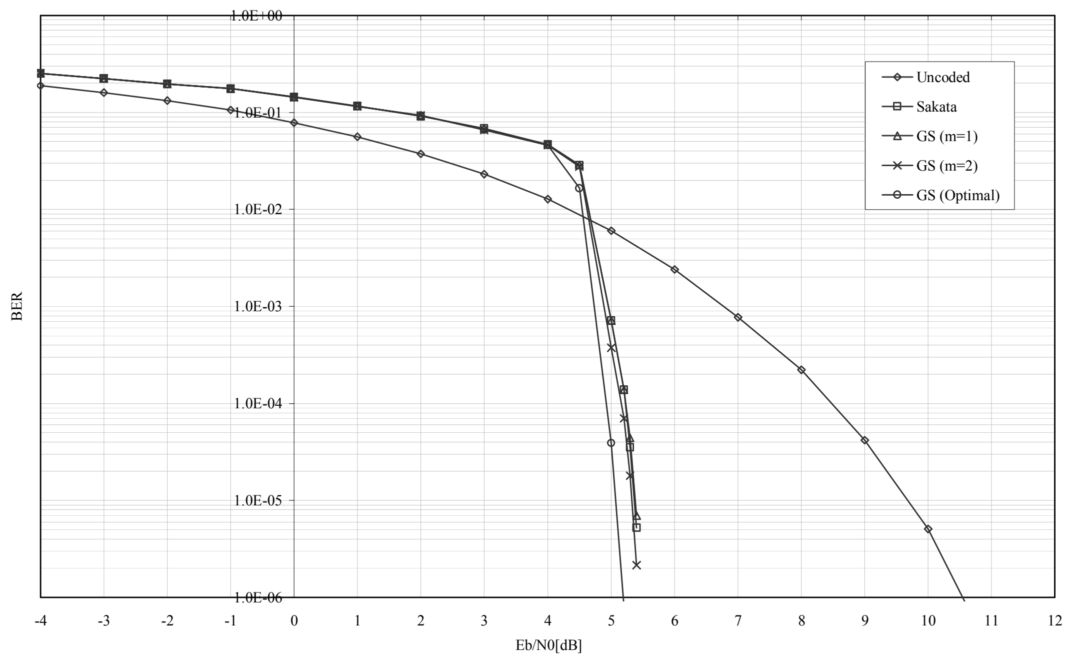


(a) over AWGN channel

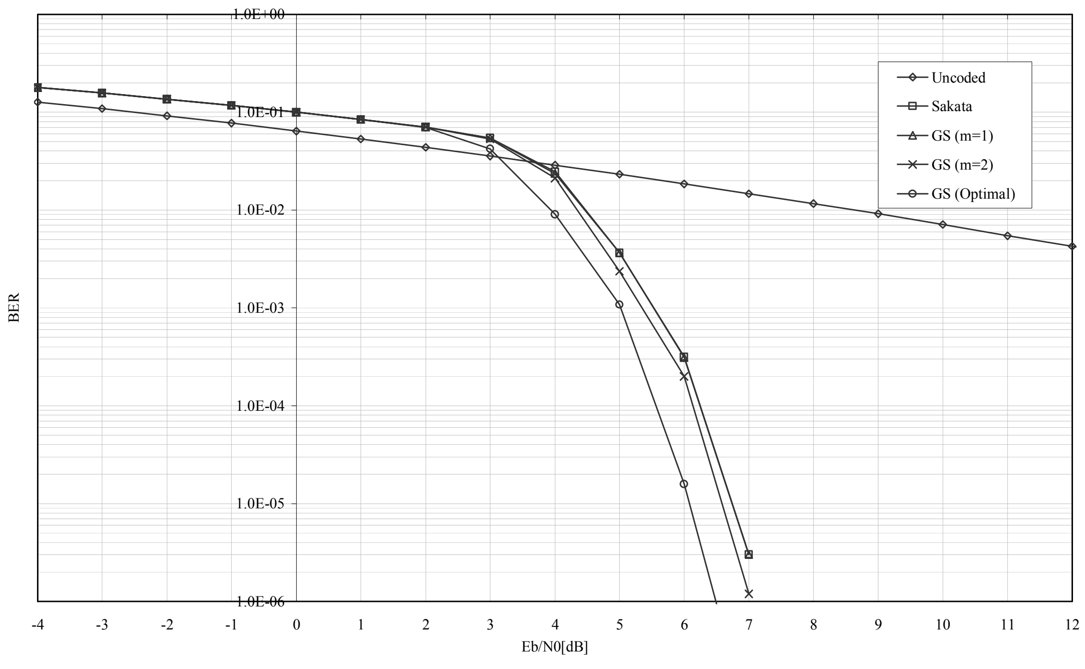


(b) over Rayleigh fading channel

Figure 6.7 Hard-decision list decoding performance of Hermitian code (512, 153, 332)



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 6.8 Hard-decision list decoding performance of Hermitian code (512, 289, 196)

Table 6.7 analyses the simulation results shown by Figs. 6.4 to 6.8. Equation (6.18) defines the GS algorithm's error-correction capability lower bound for Hermitian codes. During simulations, the average number of errors $\overline{\tau_m}$ that the GS algorithm is able to correct in order to achieve the corresponding performance is measured. Based on Table 6.7, it can be observed that the GS algorithm's coding gains grow with interpolation multiplicity *m* and they are especially significant over the Rayleigh fading channel. For example, GS decoding Hermitian code (64, 19, 40) with multiplicity *m* = 2 can achieve a 1.42 dB coding gain on the Rayleigh fading channel. The GS algorithm can achieve more significant coding gains for low rate codes, but this is at the higher expense of decoding complexity. For example, comparing

Hermitian codes	Interpolation multiplicity	$\overline{\tau_m}$	Coding gains (dB)	
			AWGN	Rayleigh fading
(64, 19, 40)	$m = 1$	20	0.17	0.71
	$m = 2$	21	0.33	1.42
	optimal ($m = 17$)	$\tau_{GS} = 24$	0.91	3.30
(64, 29, 30)	$m = 1$	14	0.01	0.10
	$m = 2$	15	0.15	1.0
	optimal ($m = 35$)	$\tau_{GS} = 17$	0.50	2.05
(64, 39, 20)	$m = 1$	9	0.10	0.01
	$m = 2$	10	0.30	0.94
	optimal ($m = 11$)	$\tau_{GS} = 10$	0.30	0.94
(512, 153, 332)	$m = 1$	167	0.05	0.16
	$m = 2$	184	0.40	0.88
	optimal ($m = 213$)	$\tau_{GS} = 208$	0.88	1.84
(512, 289, 196)	$m = 1$	97	0.01	0.01
	$m = 2$	99	0.08	0.16
	optimal ($m = 93$)	$\tau_{GS} = 109$	0.32	0.72

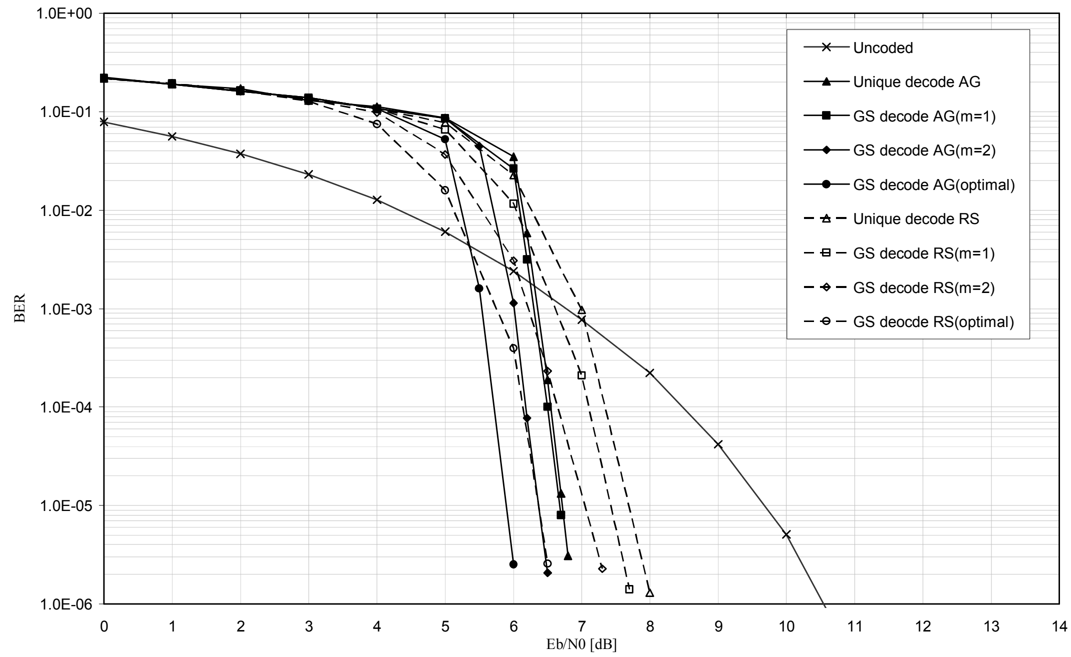
Table 6.7 Simulation results (Figs. 6.4 to 6.8) analysis

Hermitian codes (512, 153, 332) and (512, 289, 196), more significant coding gains can be achieved for the former. However, according to Table 6.6, GS decoding Hermitian code (512, 153, 332) has higher decoding complexity because there are more polynomials being initialised at the beginning to take part in the iterative interpolation. The same result can also be found by comparing Hermitian codes defined in GF(16).

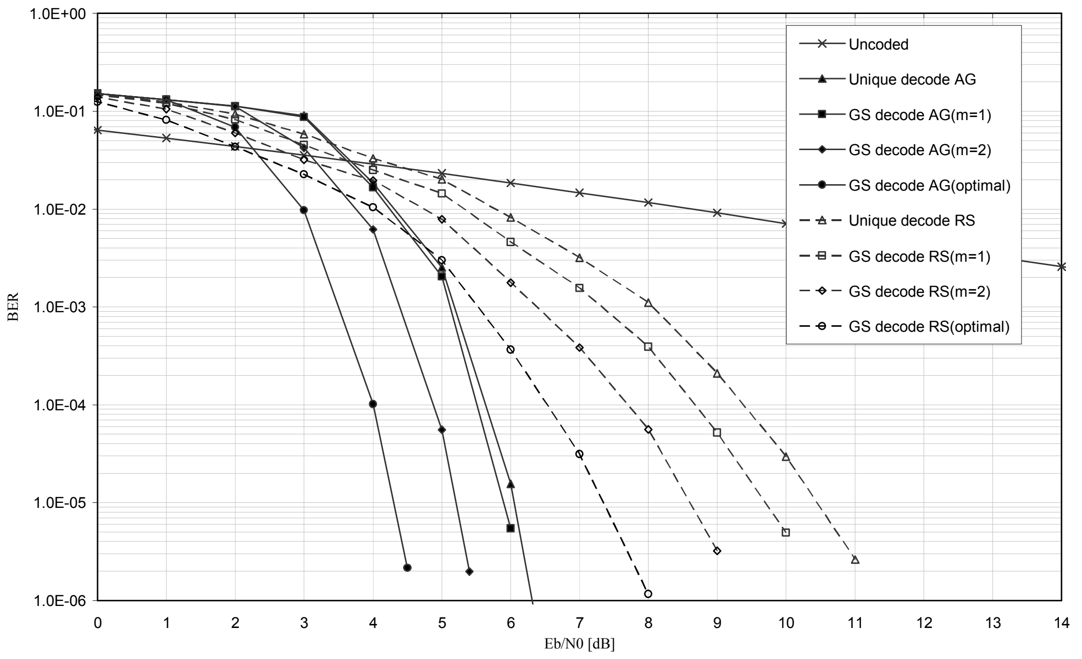
6.7.2 Comparison with Reed-Solomon Codes

By applying the list decoding algorithm, this section presents Hermitian codes performance comparisons with Reed-Solomon codes.

First, comparisons are made with a Reed-Solomon code which is defined over the same finite field. Fig. 6.9 present the comparison of Hermitian code (512, 153, 332) and Reed-Solomon code (63, 19, 45), both of which are defined in GF(64) and have code rate 0.3. This comparison result is published in the author's paper [67].



(a) over AWGN channel

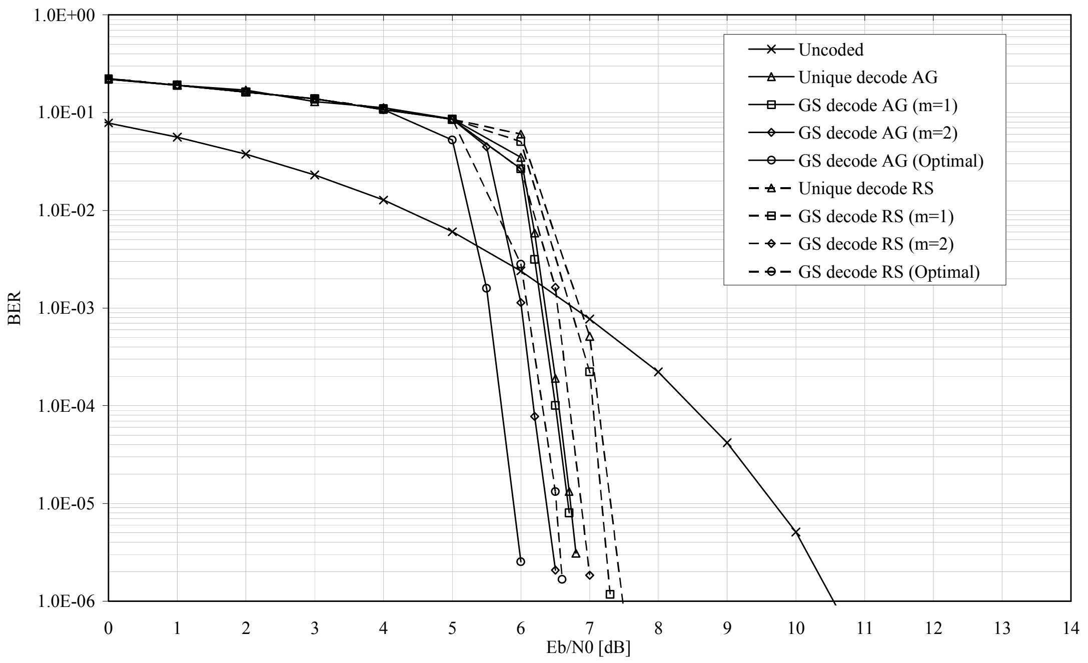


(b) over Rayleigh fading channel

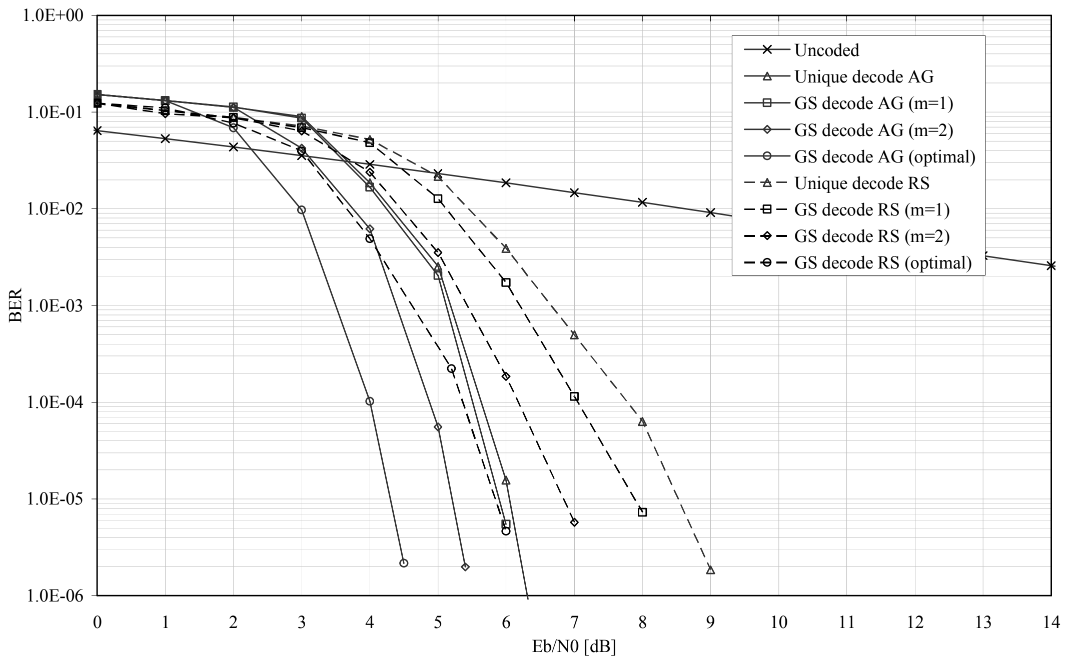
Figure 6.9 Hard-decision list decoding performance comparison of Hermitian code (512, 153, 332) and Reed-Solomon code (64, 19, 45)

Comparing the list decoding performance with respect to a certain multiplicity *m* at BER = 10^{-5} between Hermitian and Reed-Solomon codes, the Hermitian code can always outperform the Reed-Solomon code. This is because over the same finite field, longer Hermitian codes with larger minimum distances can be generated. This longer code can correct larger number of errors within one code word block, resulting performance advantage over Reed-Solomon codes. According to Fig. 6.9(a), over the AWGN channel, Hermitian code (512, 153, 332) has 0.7 dB, 0.65 dB and 0.5 dB coding gains over Reed-Solomon code (63, 19, 45) with *m* = 1, 2 and optimal, respectively. According to Fig. 6.9(b), the coding gains over the Rayleigh fading channel are more significant and are 3.7 dB, 3.3 dB and 2.9 dB with respect to *m* = 1, 2 and optimal.

Second, comparisons are made between Hermitian codes and Reed-Solomon codes which are defined in a larger finite field. Two comparison schemes are proposed: Hermitian code (512, 153, 332) compared with Reed-Solomon code (255, 76, 180), both of which have code rate 0.3 and Hermitian code (512, 289, 196) compared with Reed-Solomon code (255, 144, 112), both of which have code rate 0.56. The

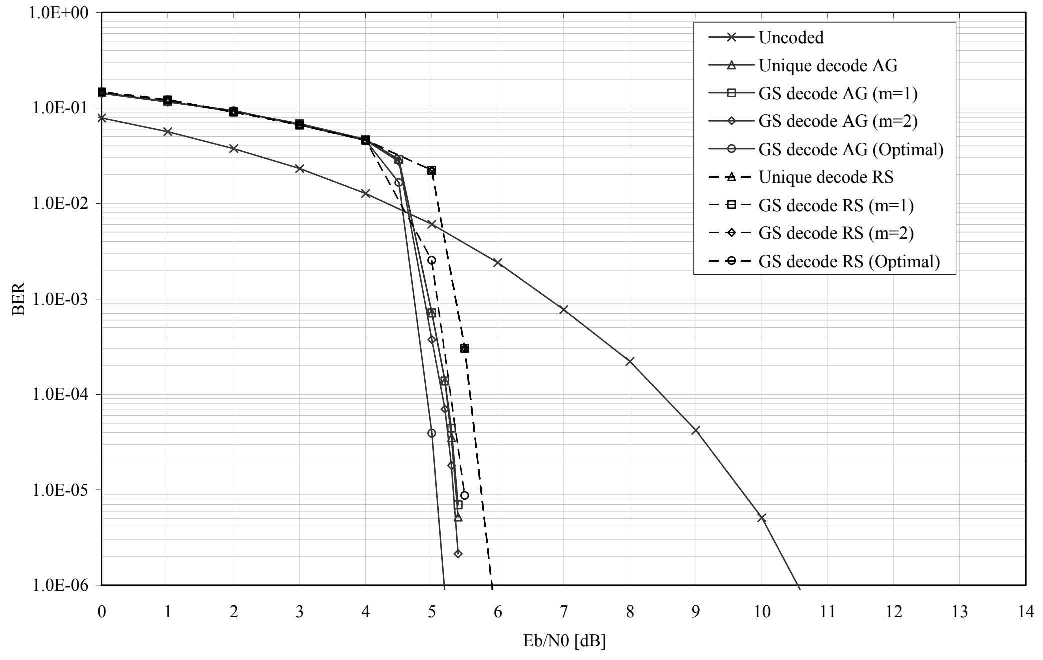


(a) over AWGN channel

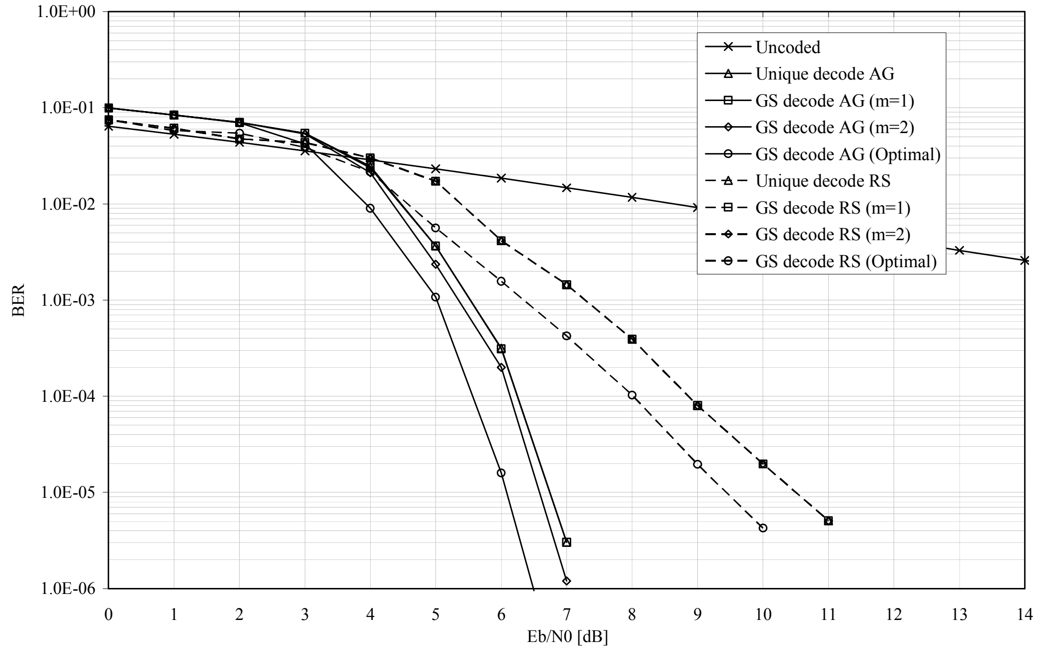


(b) over Rayleigh fading channel

Figure 6.10 Hard-decision list decoding performance comparison of Hermitian code (512, 153, 332) and Reed-Solomon code (255, 76, 180)



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 6.11 Hard-decision list decoding performance comparison of Hermitian code (512, 289, 196) and Reed-Solomon code (255, 144, 112)

Hermitian codes are defined in GF(64) while the Reed-Solomon codes are defined in GF(256). This comparison result is later published in the author's paper [68]. Notice that over the Rayleigh fading channel, 100 × 512 and 50 × 255 block interleavers are applied for Hermitian codes and Reed-Solomon codes respectively. Figs. 6.10 and 6.11 show the comparison results.

Comparisons are made by measuring the Hermitian codes' coding gains over Reed-Solomon codes at a BER equal to 10^{-5} using the same decoding parameter (multiplicity m). Based on the performance of Fig. 6.10 and 6.11, the coding gains that Hermitian codes are able to achieve over Reed-Solomon codes are summarised in Table 6.8.

Decoding Algorithms	Hermitian (512, 153, 332) / RS (255, 76, 180)		Hermitian (512, 289, 196) / RS (255, 144, 112)	
	AWGN	Rayleigh	AWGN	Rayleigh
Unique	0.62 dB	2.54 dB	0.38 dB	3.85 dB
GS (m = 1)	0.46 dB	2.08 dB	0.38 dB	3.85 dB
GS (m = 2)	0.54 dB	1.69 dB	0.46 dB	4.08 dB
GS (Optimal)	0.62 dB	1.62 dB	0.46 dB	3.46 dB

Table 6.8 Simulation results (Figs. 6.10 to 6.11) analysis

From Table 6.8's analysis, it can be seen that Hermitian codes can also outperform Reed-Solomon codes defined in a larger finite field using both the unique decoding algorithm and the GS algorithm. For Hermitian code (512, 153, 332), the GS algorithm can achieve up to 0.62 dB coding gain on the AWGN channel and 2.08 dB coding gain on the Rayleigh fading channel. For Hermitian code (512, 289, 196), the GS algorithm can achieve up to 0.46 dB and 4.08 dB coding gain on the AWGN and Rayleigh fading channel respectively. It is also interesting to notice that, for

Hermitian code (512, 153, 332), the GS algorithm with multiplicity m = 2 can outperform the optimal result that the GS algorithm can achieve for Reed-Solomon code (255, 76, 180) over both AWGN and Rayleigh fading channels, while for Hermitian code (512, 289, 196) a similar observation can be made for the GS algorithm with multiplicity m = 1. As the Reed-Solomon codes are defined over GF(256), decoding Reed-Solomon codes costs more finite field arithmetic operations than decoding Hermitian codes defined over GF(64). However, Hermitian codes still have significant performance advantages compared with Reed-Solomon codes on both AWGN and Rayleigh fading channels.

6.8 Conclusion

This chapter presented a hard-decision list decoding algorithm (GS algorithm) for Hermitian codes. Two contributions for reducing interpolation complexity have been proposed in order to improve the efficiency for list decoding of Hermitian codes. First, an efficient algorithm to determine the corresponding coefficients between the pole basis monomials and zero basis functions of a Hermitian curve was proposed. The coefficients are stored to be applied during the iterative interpolation in order to simplify the zero condition calculation of a polynomial. Then, a complexity reduction interpolation algorithm was proposed by applying a developed scheme which eliminates any unnecessary polynomials during iterations. It is shown that this scheme can improve interpolation efficiency by up to 48.83%. For the factorisation process, a generalised factorisation algorithm which can be applied to both Hermitian codes and Reed-Solomon codes was proposed. Applying the efficiency improved interpolation algorithm and a generalised factorisation algorithm, list decoding performances of Hermitian codes over AWGN and Rayleigh fading channels are evaluated. Simulation results show that the GS algorithm can achieve significant coding gains over the unique decoding algorithm. The GS algorithm's coding gains increase with interpolation multiplicity and it is more significant for low rate codes. However, according to the simulation parameters, these performance advantages are at the expenses of higher decoding complexity. By applying the list decoding algorithm, simulation results on hard-decision list decoding of Hermitian code were compared with Reed-Solomon codes. It is shown that, Hermitian codes can outperform similar code rate Reed-Solomon codes which are defined both in the same finite field and in a larger finite field.

Chapter 7

Soft-Decision List Decoding of Hermitian Codes

7.1 Introduction

Following the decoding of Hermitian codes by the Guruswami-Sudan (GS) algorithm which is a hard-decision list decoding scheme, this chapter presents a soft-decision list decoding scheme for Hermitian codes. It is developed based on Koetter and Vardy's soft-decision scheme for Reed-Solomon codes [8] which is mentioned in Chapter 5. According to Fig. 5.1, for the soft-decision scheme, the received information probabilistic reliability matrix $\boldsymbol{\Pi}$ is obtained by the receiver instead of a hard-decision received word R. Matrix $\boldsymbol{\Pi}$ is then converted to a multiplicity matrix $\boldsymbol{M}$ based on which the interpolated polynomial is built. The following analyses given in this chapter show that the soft-decision scheme is able to produce a higher code word score [8] than the hard-decision scheme, and therefore increase the system's error-correction capability. In the discussion of the simulation results for GS decoding Hermitian codes given in Chapter 6, achieving the GS algorithm's upper bound τ_{GS} (3.32) remains almost prohibitive in practice. However, this chapter shows that the soft-decision scheme with short decoded output list can outperform GS decoding's optimal result, indicating this soft-decision scheme can achieve the previously prohibitive bound τ_{GS} with a moderate decoding complexity.

An algorithm that converts the reliability matrix $\boldsymbol{\Pi}$ to multiplicity matrix $\boldsymbol{M}$ (algorithm 5.1) is presented in this chapter, introducing a stopping rule based on the designed length of the output list. For the following interpolation and factorisation processes, algorithm 6.2 and 6.3 can be applied for implementation respectively. Again, it is shown that by realising the total number of iterations in interpolation, the interpolation complexity can be reduced by eliminating polynomials with leading order greater than the iteration number.

This chapter assesses the soft-decision scheme's performance with regard to three aspects: a complexity based comparison with the hard-decision scheme, the soft-decision scheme with long output list and the soft-decision scheme's asymptotically optimal performance. This chapter's work is presented in the author's submitted paper [69].

7.2 Prerequisite Knowledge

This section presents some important parameters for introducing the soft-decision list decoding scheme.

As mentioned in section 6.2, for decoding a (n, k) Hermitian code, parameter w_z is defined as:

$$w_z = v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{k-1}^{-1}), \text{ and } w_z > 2g - 1 \tag{7.1}$$

where ϕ_{k-1} is the kth monomial in pole basis L_w of Hermitian curve H_w and g is the genus of the curve. Based on w_z, the two parameters first defined by (5.1) and (5.2) for analysing bivariate monomial $x^a y^b$ (a, $b \in$ N) can be extended to trivariate monomials $\phi_a z^b$ ($a, b \in$ N) as:

$$N_{1,w_z}(\delta) = |\{\phi_a z^b : a, b \geq 0 \text{ and } \deg_{1,w_z}(\phi_a z^b) \leq \delta, \delta \in \mathrm{N}\}| \tag{7.2}$$

which represents the number of monomials with $(1, w_z)$-weighted degree not greater than δ and

$$\Delta_{1,w_z}(v) = \min\{\delta: N_{1,w_z}(\delta) > v, v \in \mathrm{N}\} \tag{7.3}$$

which represents the minimal value of δ that guarantees $N_{1,w_z}(\delta)$ is greater v. The following corollaries associated with (7.2) and (7.3) are proposed:

Corollary 7.1: $\Delta_{1,w_z}(v) = \deg_{1,w_z}(\phi_a z^b \mid \mathrm{ord}(\phi_a z^b) = v)$.

Proof: Based on section 6.2, monomial $\phi_a z^b$'s $(1, w_z)$-lexicographic order grows based on the growth of its $(1, w_z)$-weighted degree. Up to monomial $\phi_a z^b$ with $\mathrm{ord}(\phi_a z^b) = v$, there are $v + 1$ monomials with $(1, w_z)$-weighted degree not greater than $\deg_{1,w_z}(\phi_a z^b \mid \mathrm{ord}(\phi_a z^b) = v)$. Therefore, $\deg_{1,w_z}(\phi_a z^b \mid \mathrm{ord}(\phi_a z^b) = v)$ is the minimal value that guarantees there are more than v monomials.

Corollary 7.2: $N_{1,w_z}(\delta) > \frac{(\delta - g)\delta}{2w_z}$ given $\delta > 2g - 1$, and when $\delta \to \infty$, $N_{1,w_z}(\delta) = \frac{\delta^2}{2w_z}$.

Proof: The proof is similar to the geometric arguments of Fig 5.2 given in Chapter 5. In the (1, w_z)-weighted degree table, the x-axis and y-axis represent index a of monomial ϕ_a and degree b of variable z^b and their unit distance weights 1 and w_z respectively. Each monomial $\phi_a z^b$ occupies a unit square and therefore in the table $N_{1,k-1}(\delta)$ denotes the total area occupied by monomial $\phi_a z^b$ with $\deg_{1,w_z}(\phi_a z^b) \leq \delta$. Take Table 6.1a as an example for analysis. This table can be geometrically plotted as Fig 7.1 as:

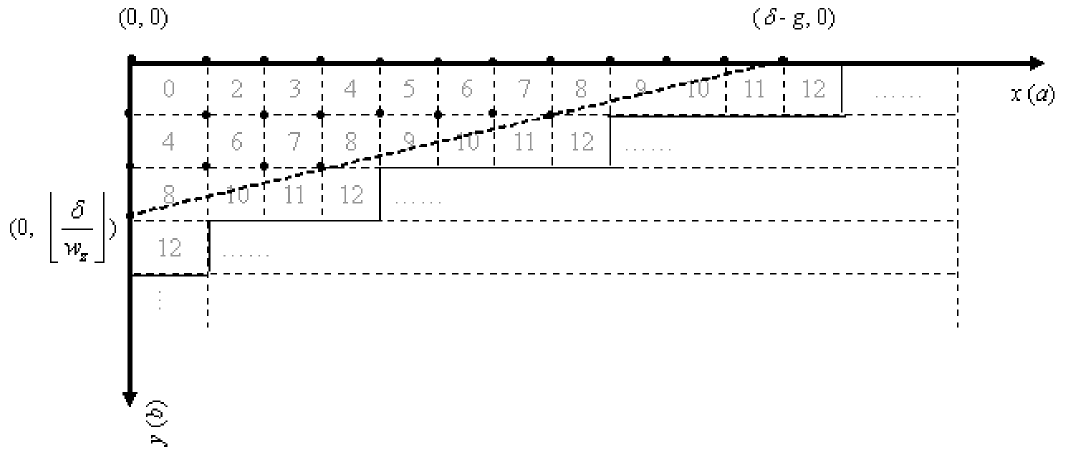


Figure 7.1 Geometric analysis of Table 6.1a

In this table, $N_{1,w_z}(\delta)$ denoted as Area 1 is enclosed by the solid line shown in the figure. The triangle defined by vertexes (0, 0), (δ - g, 0), and (0, $\left\lfloor \frac{\delta}{w_z} \right\rfloor$) has area $\frac{1}{2}(\delta - g)\left\lfloor \frac{\delta}{w_z} \right\rfloor \cong \frac{\delta(\delta - g)}{2w_z}$, which it is denoted as Area2 = $\frac{\delta(\delta - g)}{2w_z}$. From Fig 7.1, it is easy to be seen that Area1 > Area 2, and therefore $N_{1,w_z}(\delta) > \frac{\delta(\delta - g)}{2w_z}$. Also, based on the figure, it is not difficult to realise that when $\delta \to \infty$, Area1 and Area2 approach to be equal with each other and therefore $N_{1,w_z}(\delta) = \frac{\delta(\delta - g)}{2w_z}$. Since $\delta >> g$, $N_{1,w_z}(\delta) =$

$\frac{\delta^2}{2w_z}$. Take the case shown in Fig 7.1 as an example, $\delta = 12$, and $N_{1,w_z}(\delta) = N_{1,w_z}(12)$ $= 25$, while $\frac{\delta(\delta-g)}{2w_z} = \frac{12(12-1)}{2\cdot 4} = 16.5$. Therefore, $N_{1,w_z}(\delta) > \frac{\delta(\delta-g)}{2w_z}$.

7.3 Review of GS Decoding Hermitian Codes

Based on Chapter 6, given hard-decision received word $R = (r_0, r_1, \ldots, r_{n-1})$ ($r_i \in$ GF(q), $i = 0, 1, \ldots, n - 1$), n interpolated units can be formed by combining each received symbol with its respective affine point used in encoding as: (p_0, r_0), (p_1, r_1), …, (p_{n-1}, r_{n-1}). Interpolation is to build the minimal polynomial $Q_m = \sum_{a,b\in \mathrm{N}} Q_{ab}\phi_a z^b$ ($Q_{ab} \in$ GF(q)) which has a zero of multiplicity at least m over these n units. According to section 6.3, to have a zero of multiplicity m over unit (p_i, r_i), Q_m's coefficients Q_{ab} should satisfy [44]:

$$\sum_{a,b\geq\beta} Q_{ab}\binom{b}{\beta}\gamma_{a,p_i,\alpha}r_i^{b-\beta} = 0,\ \forall\ \alpha, \beta \in \mathrm{N} \text{ and } \alpha+\beta<m \quad (7.4)$$

There are in total $C_m = n\binom{m+1}{2}$ zero condition constraints (7.4) to coefficients Q_{ab}. To build polynomial Q_m, an iterative polynomial construction algorithm [9, 44, 65] can be applied.

If f is the transmitted message polynomial (3.20), we define:

$$\Lambda(f, R) = |\{i \mid f(p_i) = r_i, r_i \in R\}| \quad (7.5)$$

as the number of uncorrupted received symbols in the hard-decision received word R. Based on lemma 6.1, the total zero orders that $Q_m(x, y, f)$ has over all the interpolated units is: $\sum_{i=0}^{n-1} v_{p_i}(Q_m(x,y,f)) \geq m\ \Lambda(f, R)$ (6.14). $m\ \Lambda(f, R)$ is defined as the score of code word $\bar{c}$ as:

$$S_m(\bar{c}) = m\ \Lambda(f, R) \quad (7.6)$$

As $f \in F_q^{w_z}[x, y]$, then $v_{p_\infty}(Q_m(x, y, f)^{-1}) = \deg_{1,w_z}(Q_m(x, y, f)) = \deg_{1,w_z}(Q_m(x, y, z))$.

Hence, if $S_m(\bar{c}) > \deg_{1,w_z}(Q_m(x, y, z))$, the total zero orders of polynomial $Q_m(x, y, f)$ is greater than its pole order as: $\sum_{i=0}^{n-1} v_{p_i}(Q_m(x, y, f)) > v_{p_\infty}(Q_m(x, y, f)^{-1})$. According to theorem 6.2, message polynomial f can be found out by determining polynomial Q_m's z roots. It results the following corollary for successful list decoding.

Corollary 7.3: If the score of code word $\bar{c}$ is greater than the (1, w_z)-weighted degree of interpolated polynomial Q_m:

$$S_m(\bar{c}) > \deg_{1,w_z}(Q_m(x, y, z)) \tag{7.7}$$

then $Q_m(x, y, f) = 0$ or $z - f \mid Q_m$.

Based on corollary 7.3, it can be seen that large enough code word score is necessary to guarantee successful list decoding.

7.4 Soft-Decision List Decoding

Based on Koetter and Vardy's soft-decision scheme, this section develops a soft-decision list decoding algorithm for Hermitian codes. In the receiver, a reliability matrix $\boldsymbol{\Pi}$ of the received information is obtained instead of a hard-decision received word R. $\boldsymbol{\Pi}$ is then converted to a multiplicity matrix $\boldsymbol{M}$, based on which the interpolated polynomial is built. It is shown that the soft-decision code word score will be increased over hard-decision and consequently so will the list decoding system's error-correction capability. The algorithm that converts $\boldsymbol{\Pi}$ to $\boldsymbol{M}$ is presented with introducing a stopping rule based on a designed length of output list.

7.4.1 Reliability Information

As Chapter 5, the channel is assumed to be memoryless with input alphabet $\chi \in$ GF(q) and output alphabet $\Re \in$ GF(q), both of which are random variables from GF(q). χ is uniformly distributed over GF(q) = (ρ_0, ρ_1, …, ρ_{q-1}). In the soft-decision list decoder, the posteriori transition probability Pr($\chi = \rho_i \mid \Re = \iota_j$) can be obtained from equation (5.3) as: $\Pr(\chi = \rho_i \mid \Re = \iota_j) = \dfrac{p(\iota_j \mid \rho_i)}{\sum_{\rho \in GF(q)} p(\iota_j \mid \rho)}$. Pr($\chi = \rho_i \mid \Re = \iota_j$) represents the probability that code word $\chi = \rho_i$ was transmitted given $\Re = \iota_j$ is observed as a received word. Under continuous channel, $p(\cdot \mid \rho)$ is the probability-density function and $\Re$ is continuous. Under discrete channel, $p(\cdot \mid \rho)$ is the probability-mass function and $\Re$ is discrete.

As the transmitted code word can be chosen from the q finite field elements, for every random received variable ι_j, there are q posteriori transition probability values. Therefore, for the received vector $\overline{\Re} = (\iota_0, \iota_1, \ldots, \iota_{n-1})$, a $q \times n$ reliability matrix $\boldsymbol{\Pi}$ can be obtained:

$$\boldsymbol{\Pi} = \begin{bmatrix} \pi_{0,0} & \pi_{0,1} & \cdots & \cdots & \cdots & \pi_{0,n-1} \\ \pi_{1,0} & \pi_{1,1} & & & & \pi_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & \pi_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ \pi_{q-1,0} & \pi_{q-1,1} & \cdots & \cdots & \cdots & \pi_{q-1,n-1} \end{bmatrix} \quad (7.8)$$

where entry $\pi_{i,j}$ represents the probability that ρ_i was transmitted given ι_j is observed:

$$\pi_{i,j} = \Pr(\chi = \rho_i \mid \Re = \iota_j) \ (i = 0, 1, \ldots, q - 1 \text{ and } j = 0, 1, \ldots, n - 1) \quad (7.9)$$

Reliability matrix $\boldsymbol{\Pi}$ represents every possible transmitted code word symbol's transition probability. This received information's reliability matrix $\boldsymbol{\Pi}$ is then converted to a $q \times n$ multiplicity matrix $\boldsymbol{M}$, based on which the interpolated polynomial $Q_M \in F_q[x, y, z]$ is built (Note: in this chapter, Q_M denotes the soft-decision interpolated polynomial and Q_m denotes the hard-decision interpolated polynomial). The method for obtaining this reliability matrix based on channel output observation is presented in section 5.3. The difference between soft-decision list decoding of Reed-Solomon codes and Hermitian codes is in the size of the reliability matrix. For soft-decision list decoding of Reed-Solomon code, the reliability matrix $\boldsymbol{\Pi}$ (5.4) has size $q \times n$, where $n = q - 1$. For soft-decision list decoding of Hermitian code, the reliability matrix $\boldsymbol{\Pi}$ (7.8) has size $q \times n$, where $n = q^{3/2}$.

7.4.2 System Solution

The reliability matrix $\boldsymbol{\Pi}$ is then converted to the multiplicity matrix $\boldsymbol{M}$, for which algorithm 5.1 is applied. This algorithm will be presented later in this subsection with introducing a stopping rule based on the designed length of output list.

According to corollary 7.3, high code word score is necessary for successful list decoding. In this subsection, the code word score with respect to matrix $\boldsymbol{M}$ is analysed so as to present the system solution for this soft-decision list decoder. It is shown that the soft-decision scheme provides a higher code word score than the hard-decision scheme.

The resulting multiplicity matrix $\boldsymbol{M}$ can be written as:

$$\boldsymbol{M} = \begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & \cdots & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & & & & m_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & m_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ m_{q-1,0} & m_{q-1,1} & \cdots & \cdots & \cdots & m_{q-1,n-1} \end{bmatrix} \tag{7.10}$$

where entry $m_{i,j}$ represents the multiplicity for unit (p_j, ρ_i). Different to the multiplicity matrix $\boldsymbol{M}$ (5.8) with size $q \times q - 1$, multiplicity matrix $\boldsymbol{M}$ (7.10) has size $q \times q^{3/2}$. Interpolation is to build the minimal polynomial $Q_M \in F_q[x, y, z]$ which has a zero of multiplicity at least $m_{i,j}$ ($m_{i,j} \neq 0$) over all the associated units (p_j, ρ_i). Following from (7.4), with respect to interpolated unit (p_j, ρ_i), Q_M's coefficients Q_{ab} should satisfy:

$$\sum_{a,b\geq\beta} Q_{ab}\binom{b}{\beta}\gamma_{a,p_j,\alpha}\rho_i^{\,b-\beta} = 0,\ \forall\ \alpha, \beta \in \mathrm{N} \text{ and } \alpha+\beta< m_{i,j} \tag{7.11}$$

For this soft-decision interpolation, the number of interpolated units covered by Q_M is:

$$|\{ m_{i,j} \neq 0 \mid m_{i,j} \in \boldsymbol{M}, i = 0, 1, \ldots, q - 1 \text{ and } j = 0, 1, \ldots, n - 1\}| \tag{7.12}$$

and the cost C_M of multiplicity matrix $\boldsymbol{M}$ is:

$$C_M = \frac{1}{2}\sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}(m_{i,j}+1) \tag{7.13}$$

which represents the number of constraints (7.11) to Q_M's coefficients Q_{ab}. They can be imposed by the iterative polynomial construction algorithm [9, 44, 65] in C_M iterations. Notice that equation (7.12) and (7.13) has the same expression as equation (5.9) and (5.11) respectively, except in equations (7.12) and (7.13) $n = q^{3/2}$.

Based on lemma 6.1, the following units' multiplicities will contribute to the code word score: (p_0, c_0), (p_1, c_1), …, and (p_{n-1}, c_{n-1}). Referring to the multiplicity matrix (7.10), the interpolated polynomial Q_M can be explained as passing through these units with multiplicity at least $m_0 = m_{i,0}$ $(\rho_i = c_0)$, $m_1 = m_{i,1}$ $(\rho_i = c_1)$, …, and $m_{n-1} = m_{i,n-1}$ $(\rho_i = c_{n-1})$ respectively. If $f \in F_q^{w_z}[x, y]$ is the transmitted message polynomial that $f(p_i) = c_i$, the total zero order of $Q_M(x, y, f)$ over units $\{(p_0, c_0), (p_1, c_1), \ldots, (p_{n-1}, c_{n-1})\}$ is at least:

$$m_0 + m_1 + \cdots + m_{n-1} = \sum_{j=0}^{n-1}\{m_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} \tag{7.14}$$

And therefore, the code word score $S_M(\bar{c})$ with respect to multiplicity matrix $\boldsymbol{M}$ is:

$$S_M(\bar{c}) = \sum_{j=0}^{n-1}\{m_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} \tag{7.15}$$

If $S_M(\bar{c}) > \deg_{1,w_z}(Q_M(x, y, z))$, then $\sum_{i=0}^{n-1} v_{p_i}(Q_M(x, y, f)) > v_{p_\infty}(Q\ (x, y, f)^{-1})$ and $Q_M(x, y, f) = 0$. f can be found out by determining $Q_M(x, y, z)$'s z roots. It results the following corollary for successful soft-decision list decoding.

Corollary 7.4: If the code word score with respect to multiplicity matrix $\boldsymbol{M}$ is greater than the interpolated polynomial Q_M's $(1, w_z)$-weighted degree, as:

$$S_M(\bar{c}) > \deg_{1,w_z}(Q_M(x, y, z)) \tag{7.16}$$

then $Q_M(x, y, f) = 0$ or $z - f \mid Q_M(x, y, z)$.

To compare the soft-decision's code word score $S_M(\bar{c})$ with the hard-decision's code word score $S_m(\bar{c})$, denote the index of the maximal element in each column of $\boldsymbol{\Pi}$ as:

$$i_j = \text{index}\,(\max\{\pi_{i,j} \mid i = 0, 1, \ldots, q\text{-}1\}) \tag{7.17}$$

that $\pi_{i_j,j} > \pi_{i,j}$ ($i \neq i_j$). The hard-decision received word R can be written as:

$$R = (r_0, r_1, \ldots, r_{n\text{-}1}) = (\rho_{i_0}, \rho_{i_1}, \ldots, \rho_{i_{n-1}}) \tag{7.18}$$

For hard-decision list decoding, in the multiplicity matrix (7.10), only those entries that correspond to the reliability value $\pi_{i_j,j}$ will be assigned a multiplicity as $m_{i_j,j} = m$, and therefore the score in (7.6) of hard-decision can also be written with respect to multiplicity matrix $\boldsymbol{M}$ as:

$$S_m(\bar{c}) = S_M(\bar{c}) = \sum_{j=0}^{n-1}\{m_{i_j,j} \mid \rho_{i_j} = c_j\} \tag{7.19}$$

Comparing (7.15) and (7.19), the soft-decision list decoder gains its improvements by increasing its code word score. This is done by increasing the total number of interpolated units (7.12) so that it can increase the possibility of covering more interpolated units which include the corresponding code word symbols.

If the $(1, w_z)$-weighted degree of interpolated polynomial Q_M is δ^*, based on (7.2), Q_M has at most $N_{1,w_z}(\delta^*)$ nonzero coefficients. The interpolation procedure generates a system of C_M linear equations of type (7.11). The system will be solvable if [8]:

$$N_{1,w_z}(\delta^*) > C_M \tag{7.20}$$

Based on (7.3), in order to guarantee the solution, the $(1, w_z)$-weighted degree δ^* of the interpolated polynomial Q_M should be large enough so that:

$$\deg_{1,w_z}(Q_M(x, y, z)) = \delta^* = \Delta_{1,w_z}(C_M) \tag{7.21}$$

Therefore, based on corollary 7.4, given the soft-decision code word score (7.15) and the (1, w_z)-weighted degree of the interpolated polynomial Q_M (7.21), message polynomial f can be found out if:

$$S_M(\bar{c}) > \Delta_{1,w_z}(C_M) \tag{7.22}$$

The factorisation output list contains the z roots of polynomial Q_M. Therefore, the maximal length of output list l_M should be equal to polynomial Q_M's z degree ($\deg_z Q_M$) as:

$$l_M = \deg_z(Q_M(x, y, z)) = \left\lfloor \frac{\deg_{1,w_z}(Q_M(x, y, z))}{w_z} \right\rfloor = \left\lfloor \frac{\Delta_{1,w_z}(C_M)}{w_z} \right\rfloor \tag{7.23}$$

During converting matrix $\boldsymbol{\Pi}$ to matrix $\boldsymbol{M}$, based on a designed length of output list l, algorithm 5.1 will stop once l_M is greater than l. According to corollary 7.1, $\Delta_{1,w_z}(C_M)$ can be determined by finding the monomial with (1, w_z)-lexicographic order C_M as:

$$\Delta_{1,w_z}(C_M) = \deg_{1,w_z}(\phi_a z^b \mid \mathrm{ord}(\phi_a z^b) = C_M) \tag{7.24}$$

Therefore, in order to assess the soft-decision list decoding algorithm's performance with a designed length of output list l, a large enough value is set for s when initialising algorithm 5.1. In the algorithm, after step (v), we can determine the cost C_M (7.13) of the updated matrix $\boldsymbol{M}$ and apply (7.24) to determine $\Delta_{1,w_z}(C_M)$. Then the maximal length of output list l_M can be determined by (7.23). Stop algorithm 5.1 once l_M is greater than l and output the updated matrix $\boldsymbol{M}$.

Here algorithm 5.1 is again presented with introducing this stopping rule. This presented version is used in the author's practical simulations.

Algorithm 7.1: Convert reliability matrix $\boldsymbol{\Pi}$ to multiplicity matrix $\boldsymbol{M}$.

Input: Reliability matrix $\boldsymbol{\Pi}$, a high enough desired value of $s = \sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}$, and designed output length l.

```
Initialisation: Set $\boldsymbol{\Pi}^* = \boldsymbol{\Pi}$ and $q \times n$ all-zero multiplicity matrix $\boldsymbol{M}$
(i): While ($s > 0$ or $\lfloor l_M \rfloor < l$) {
(ii): Find the maximal entry $\pi^*_{i,j}$ in $\boldsymbol{\Pi}^*$ with position $(i, j)$
(iii): Update $\pi^*_{i,j}$ in $\boldsymbol{\Pi}^*$ as $\pi^*_{i,j} = \frac{\pi_{i,j}}{m_{i,j} + 2}$
(iv): Update $m_{i,j}$ in $\boldsymbol{M}$ as $m_{i,j} = m_{i,j} + 1$
(v): $s = s - 1$
(vi): For the updated $\boldsymbol{M}$, calculate its interpolation cost $C_M$ by (7.13)
(vii): Determine $\Delta_{1,w_z}(C_M)$ by (7.24)
(viii): Calculate $l_M$ by (7.23)
}
```

Again, this algorithm gives priority to those interpolated points which correspond to a higher reliability values $\pi_{i,j}$ to be assigned with a higher multiplicity values $m_{i,j}$. For example, if $\pi_{i_1j_1} < \pi_{i_2j_2}$, then $m_{i_1j_1} \leq m_{i_2j_2}$.

7.5 Complexity reduction Interpolation and Factorisation

To implement the following interpolation and factorisation processes, algorithm 6.2 and 6.3 are applied respectively. Algorithm 6.2 produces the interpolated polynomial $Q_M(x, y, z)$ which has a zero of multiplicity at least $m_{i,j}$ ($m_{i,j} \neq 0$) over all the associated interpolated units (p_j, ρ_i). This section presents some modifications for algorithm 6.2 for the soft-decision interpolation process while the factorisation process remains the same.

As mentioned in Chapter 6, to efficiently implement algorithm 6.2, algorithm 6.1 needs to be applied first to determine all the necessary corresponding coefficients $\gamma_{a,p_j,\alpha}$ (with respect to affine point p_j). The interpolated polynomial's $(1, w_z)$-weighted degree upper bound can be determined by (7.21) based on knowing the iteration number C_M. Therefore, the maximum pole basis that might exist in the interpolated polynomial can be determined by: $v_{p_\infty}(\phi_{\max}^{-1}) = \deg_{1,w_z}(Q_M(x, y, z)) = \Delta_{1,w_z}(C_M)$. Algorithm 6.1 is applied to determine the corresponding coefficients $\gamma_{0,p_j,\alpha} \sim \gamma_{\max,p_j,\alpha}$ for all the affine points (p_0, p_1, …, p_{n-1}). Based on the above description of algorithm 7.1, if i_j (7.17) is the index of the maximal entry $\pi_{i,j}$ in column j of matrix $\boldsymbol{\Pi}$, then $m_{i_j,j}$ is the maximal entry in column j of matrix $\boldsymbol{M}$. With respect to affine point p_j, only corresponding coefficients $\gamma_{0,p_j,\alpha} \sim \gamma_{\max,p_j,\alpha}$ ($\alpha < m_{i_j,j}$) will be stored for use in the following interpolation process.

Based on the pre-determined corresponding coefficients, algorithm 6.2 will run through all the interpolated units (p_j, ρ_i) with interpolation multiplicity $m_{i,j}$ ($m_{i,j} \neq 0$). If l is the designed length of output list, at the beginning of algorithm 6.2, the polynomial group initialisation (6.28) should be written as:

$$G = \{Q_j = Q_{\lambda + w\delta} = y^\lambda z^\delta,\ 0 \le \lambda < w,\ 0 \le \delta \le l\} \qquad (7.25)$$

According to Chapter 6, the complexity reduction scheme [52] is also valid for the interpolation process of Hermitian codes. The chosen interpolated polynomial Q_M's leading order will not be greater than the iteration number C_M:

$$\mathrm{lod}(Q_M) \le C_M \qquad (7.26)$$

and as a result

$$\deg_{1,w_z}(Q_M(x, y, z)) \le \Delta_{1,w_z}(C_M) \qquad (7.27)$$

which indicates (7.21) is the (1, w_z)-weighted degree upper bound for the interpolated polynomial Q_M. Therefore, the complexity reduction modification (6.29) of algorithm 6.2 can be written as:

$$G = \{Q_j \mid \mathrm{lod}(Q_j) \le C_M\} \qquad (7.28)$$

in which iteration number C_M can be determined by (7.13) after the multiplicity matrix $\boldsymbol{M}$ is obtained from algorithm 7.1. With respect to interpolated unit (p_j, ρ_i), the zero condition test Δ_j performed by (6.30) is redefined by equation (7.11). Also, in the polynomial modification (6.33), x_i should be replaced by x_j which is the x-coordinate of the affine point p_j in the current interpolated unit (p_j, ρ_i). It is worthy to mention that index j needs to be distinguished when it is applied as an index for polynomials Q_j in the group and when it is applied as an index for affine point p_j. After C_M iterations, the minimal polynomial is chosen from the group G as:

$$Q_M = \min_{lod(Q_j)} (Q_j \mid Q_j \in G) \tag{7.29}$$

Fig 7.2 shows how much computational complexity can be reduced by modification scheme (7.28) for soft-decision interpolation of Hermitian code (64, 19, 40) with output length l = 2 and l = 3. Again, as no hard-decision received word is obtained in the soft-decision decoder, the complexity analysis is measured against the SNR values. For Fig 7.2, it can be seen that in high SNR value situations, complexity reduction is more significant. It can be reduced up to 21.76% when l = 3 and 15.10% when l = 2.

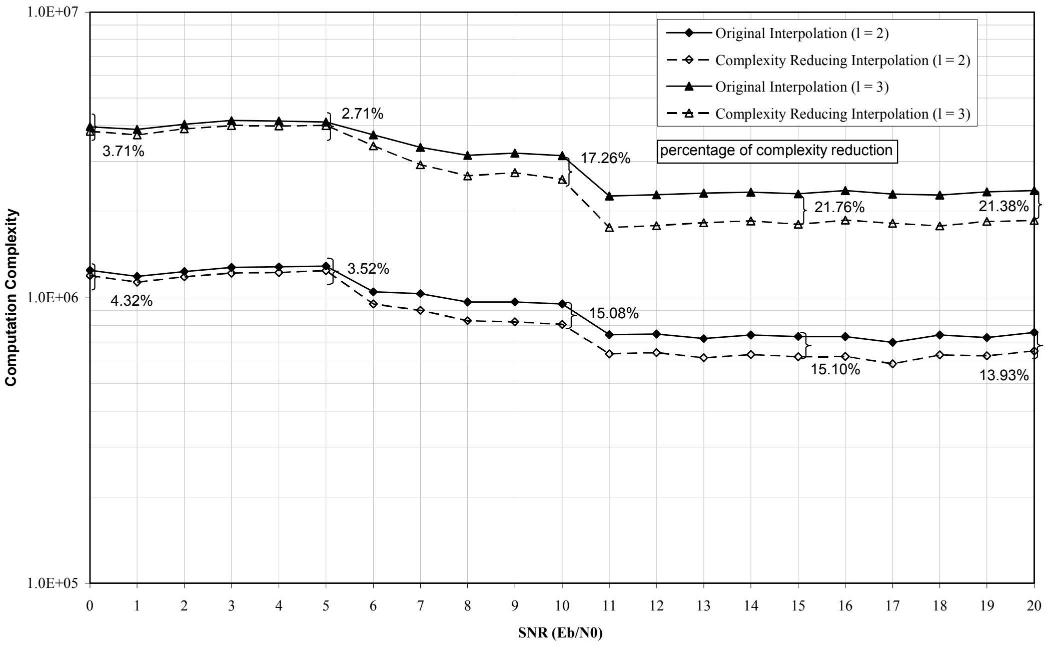


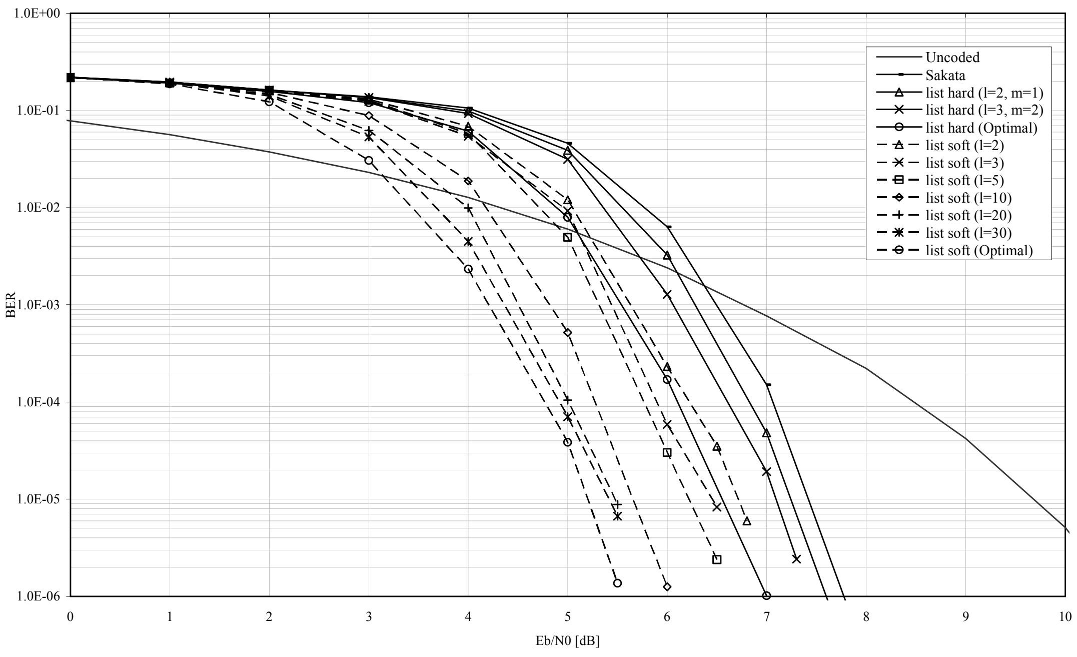
Figure 7.2 Complexity reduction analysis for soft-decision list decoding of Hermitian code (64, 19, 40)

7.6 Simulation Results Discussion

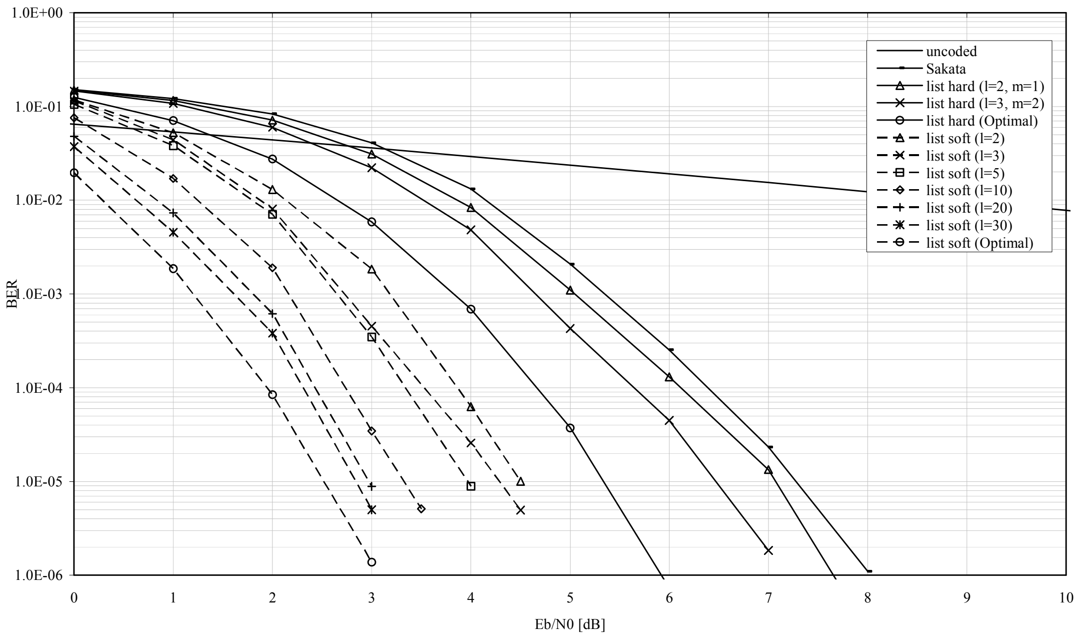
Applying the above soft-decision list decoder, this section assesses the performance for Hermitian codes defined in GF(16) and GF(64), whose hard-decision list decoding performances are presented in section 6.7. Figs 7.3, 7.4 and 7.5 present the performances of Hermitian codes (64, 19, 40), (64, 29, 30) and (64, 39, 20) respectively, while Figs 7.6 and 7.7 present the performances of Hermitian codes (512, 153, 332) and (512, 289, 196) respectively. Performance is evaluated over AWGN and Rayleigh fading channels using the QPSK modulation scheme. The Rayleigh fading channel is frequency nonselective with Doppler frequency 126.67 Hz. The fading profile is generated using Jakes' method [64]. The fading coefficients have mean value 1.55 and variance 0.60. During simulation, quasi-static fading is assumed in which the fading amplitude changes for each code word block. For combating the fading effect, 64×64 and 100×512 block interleavers are employed for codes defined in GF(16) and GF(64) respectively. Analyses of Figs 7.3 to 7.7 emphasise the following three aspects: a complexity based comparison with hard-decision decoding, soft-decision decoding performance assessment with a large length of output list and its asymptotically optimal performance.

7.6.1 Complexity Based Comparison with Hard-Decision

This subsection assesses the soft-decision decoding performance comparison with hard-decision decoding based on similar decoding complexity. This comparison is made by having the same designed length of output list l which is the main parameter that decides the decoding complexity. According to section 7.5, there are $w(l + 1)$ polynomials initialised at the beginning to take part in the iterative polynomial construction process for interpolation. According to (7.23), for a higher value of l, the interpolated polynomial should have a higher $(1, w_z)$-weighted degree which is built in a higher number of iterations (C_m for hard-decision, C_M for soft-decision). Both the number of polynomial $w(l + 1)$ and the number of iterations (C_m, C_M) significantly affect the decoding complexity.

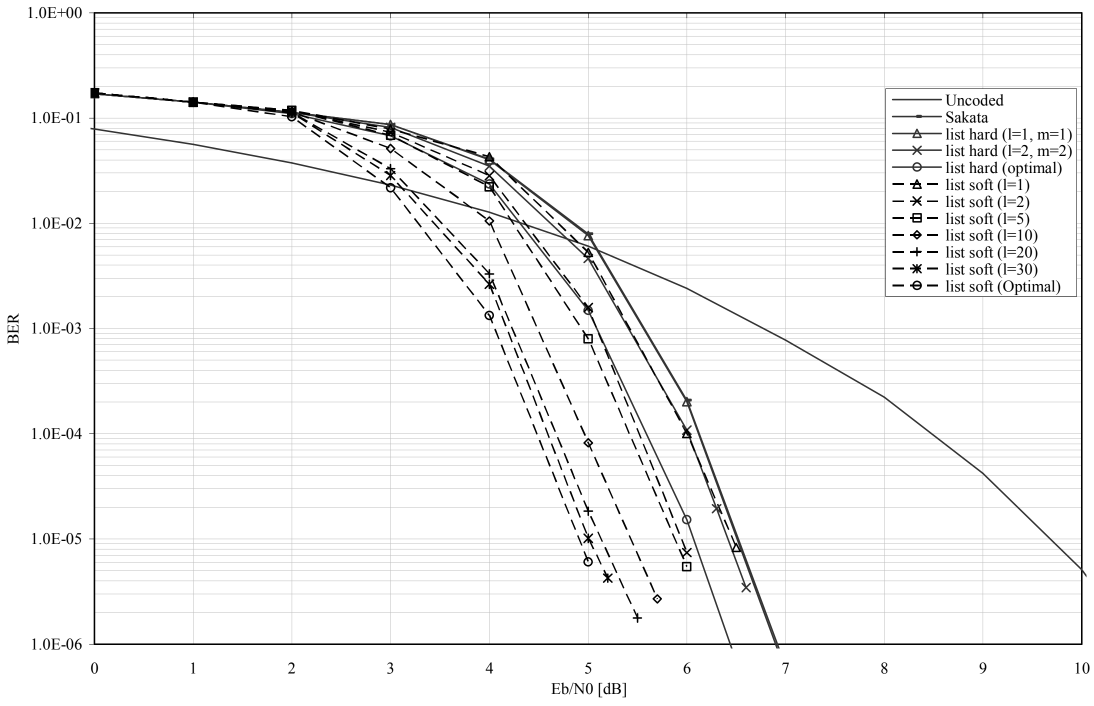


(a) over AWGN channel

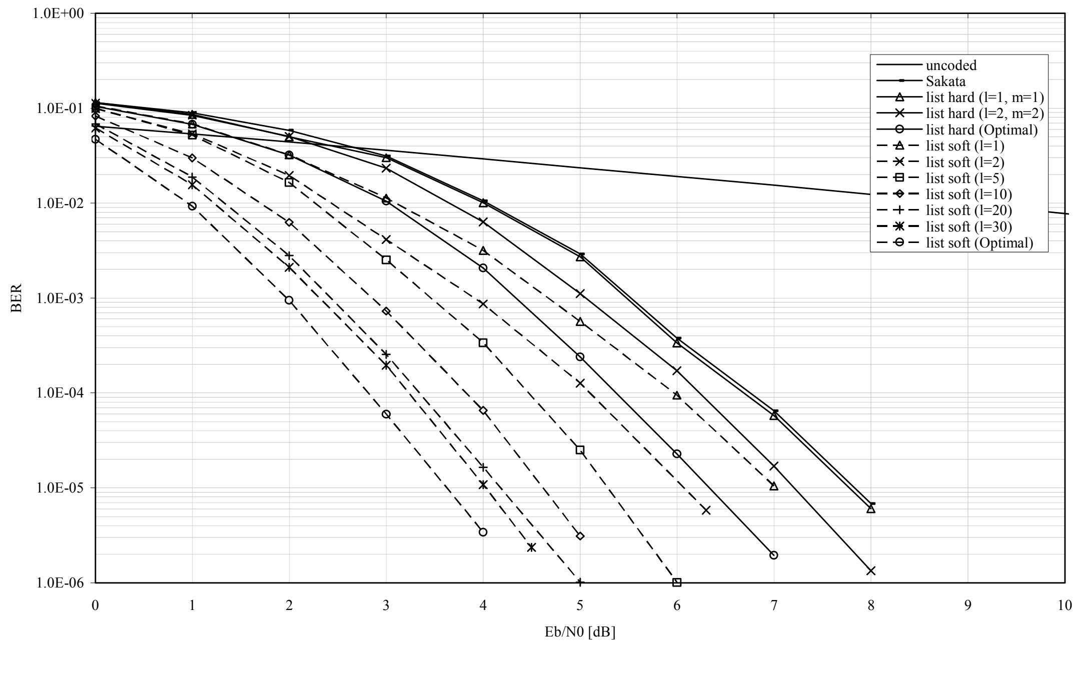


(b) over Rayleigh fading channel

Figure 7.3 Soft-decision list decoding of Hermitian code (64, 19, 40)

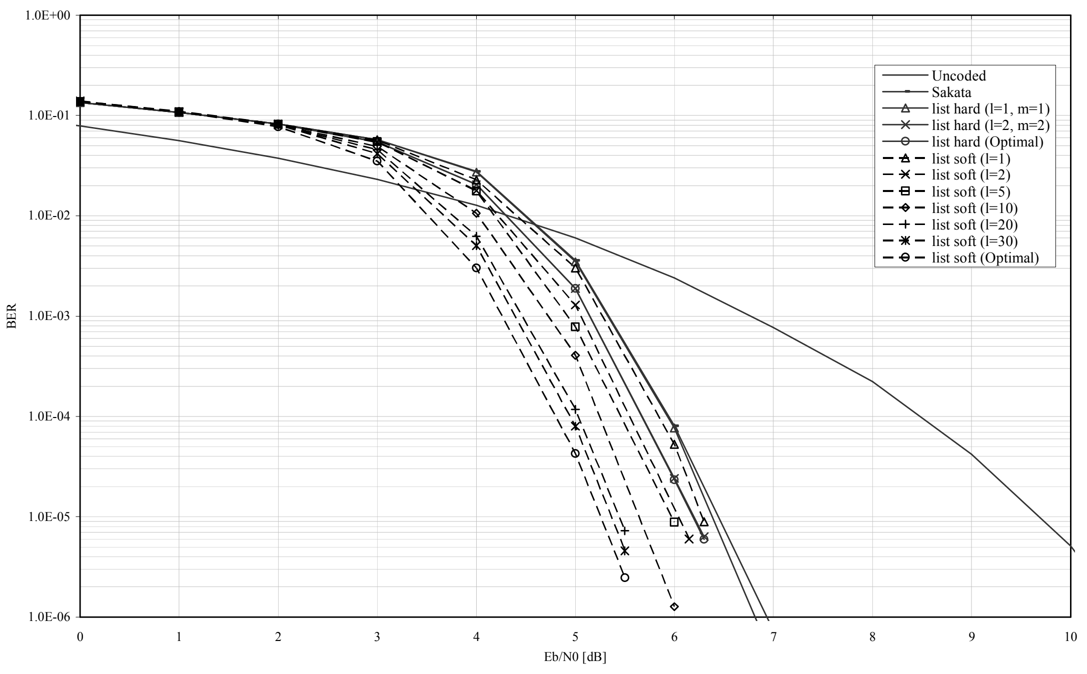


(a) over AWGN channel

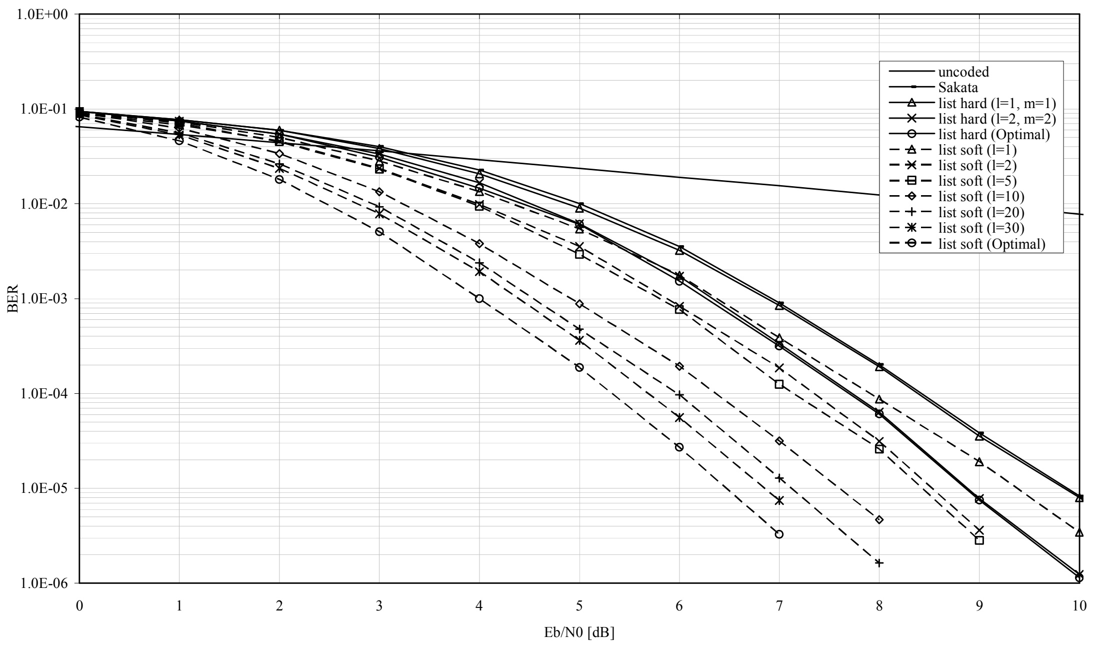


(b) over Rayleigh fading channel

Figure 7.4 Soft-decision list decoding of Hermitian code (64, 29, 30)

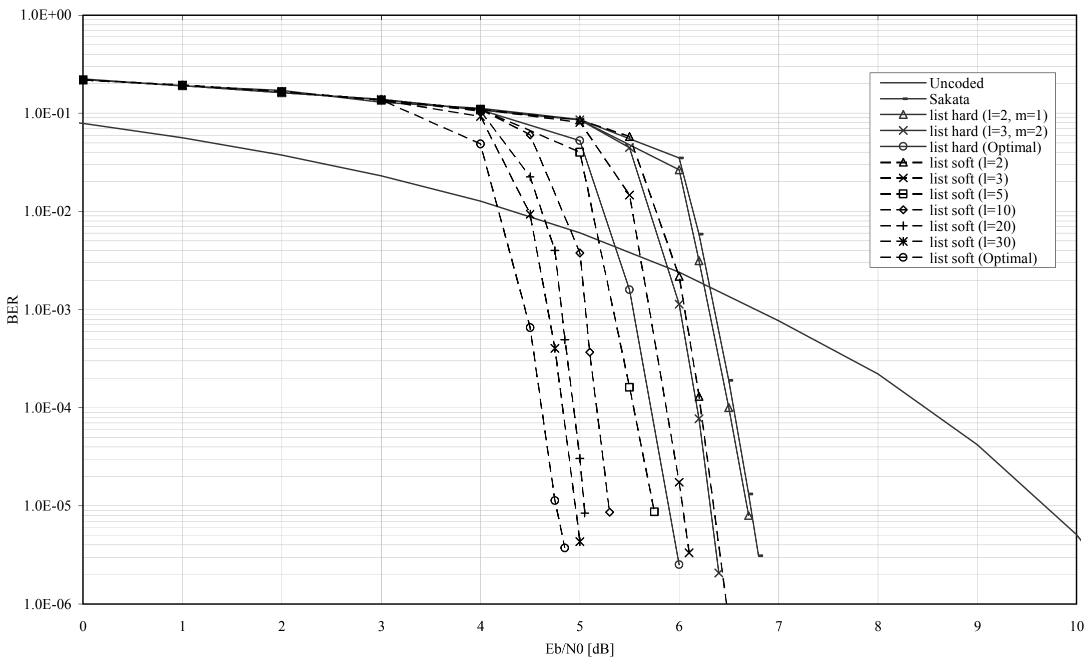


(a) over AWGN channel

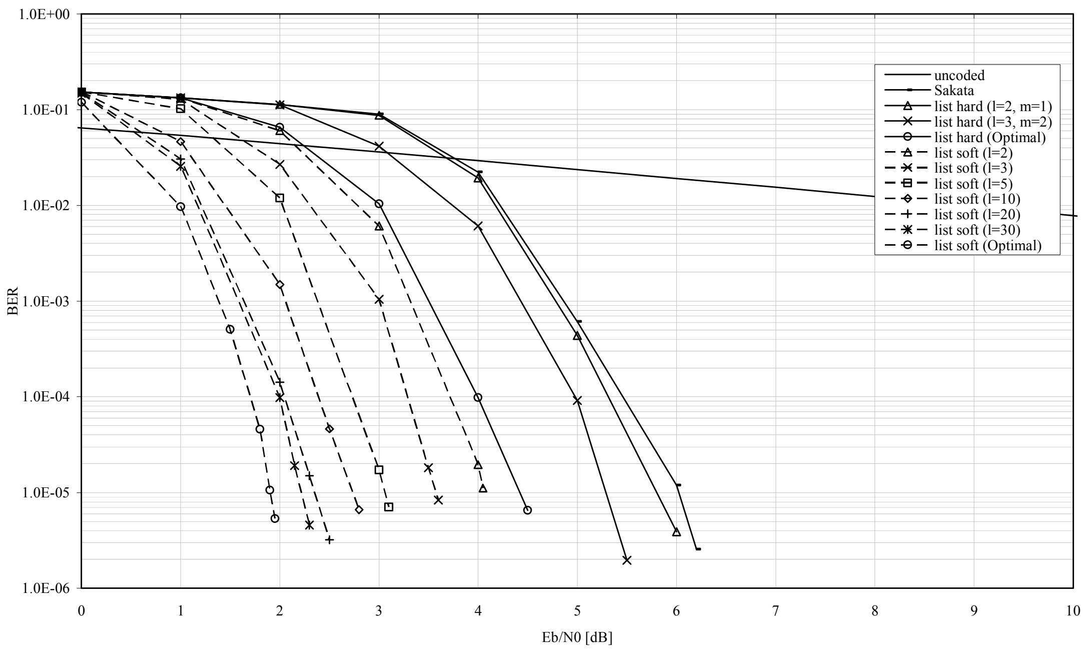


(b) over Rayleigh fading channel

Figure 7.5 Soft-decision list decoding of Hermitian code (64, 39, 20)

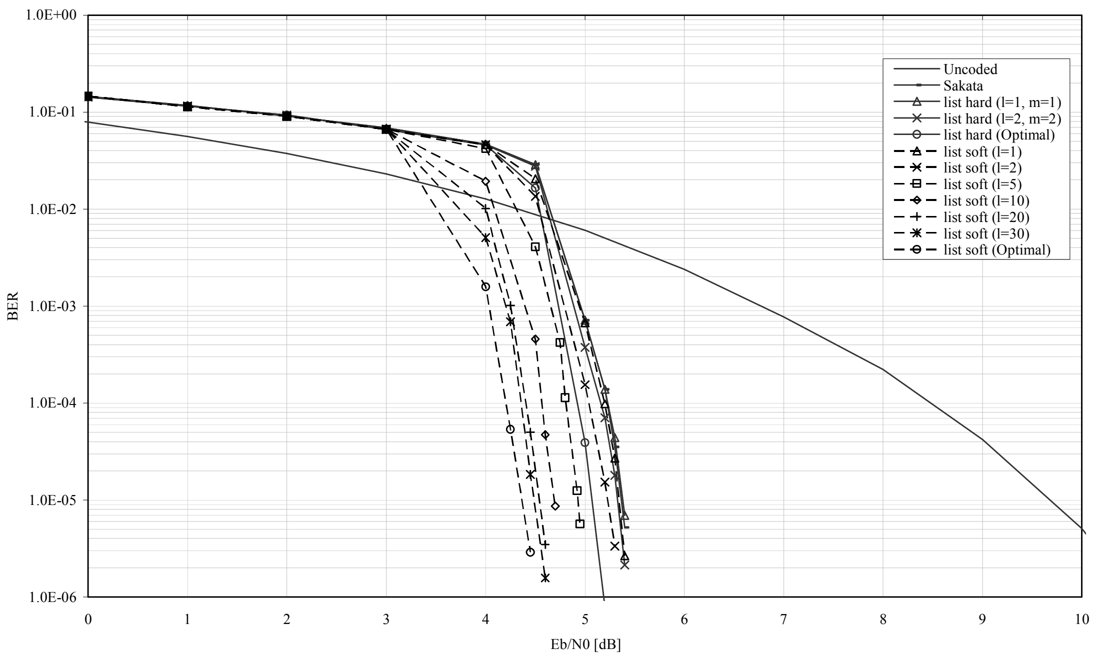


(a) over AWGN channel

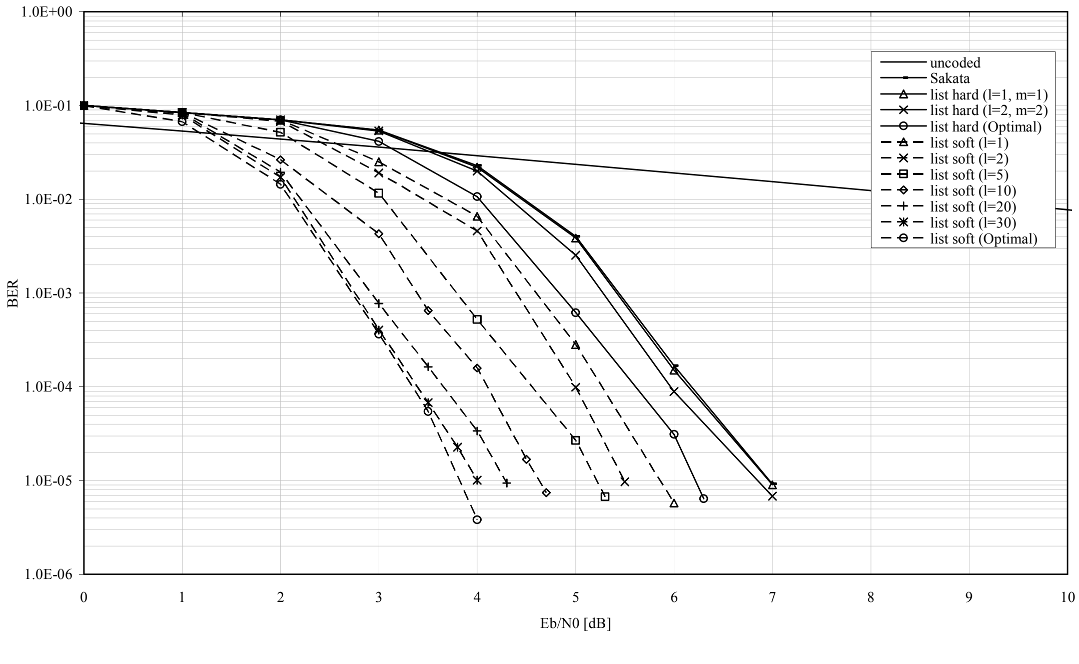


(b) over Rayleigh fading channel

Figure 7.6 Soft-decision list decoding of Hermitian code (512, 153, 332)



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 7.7 Soft-decision list decoding of Hermitian code (512, 289, 196)

Hermitian codes	Designed length of output list l	Number of polynomials $w(l + 1)$	Hard-decision iterations $C_m(m)$	Soft-decision iterations C_M
(64, 19, 40)	2	12	64 ($m = 1$)	127
	3	16	192 ($m = 2$)	218
	5	24	384 ($m = 3$)	468
(64, 29, 30)	1	8	64 ($m = 1$)	90
	3	16	192 ($m = 2$)	317
	5	24	640 ($m = 4$)	680
(64, 39, 20)	1	8	64 ($m = 1$)	120
	2	12	192 ($m = 2$)	246
	3	16	384 ($m = 3$)	418
(512, 153, 332)	2	24	512 ($m = 1$)	997
	3	32	1536 ($m = 2$)	1688
	5	48	3072 ($m = 3$)	3612
(512, 289, 196)	1	16	512 ($m = 1$)	892
	2	24	1536 ($m = 2$)	1813
	4	40	3072 ($m = 3$)	4602

Note: Iteration number C_M for soft-decision is an average value observed from simulations.

Table 7.1 Interpolation complexity comparison for soft-decision and hard-decision list decoding

Table 7.1 shows, for list decoding of these 5 Hermitian codes, based on a designed length of output list l, the number of polynomials $w(l + 1)$ and the number of iterations (C_m, C_M) for both hard-decision and soft-decision. From Table 7.1, it can be observed that based on the same value of l, C_M is higher than C_m, which indicates soft-decision list decoder has higher decoding complexity. But a significant change of iterations is due to the change of designed length *l*. From these simulation results, it can be seen that based on the same length of output list, soft-decision list decoding can outperform hard-decision. At BER of 10^{-5}, the coding gain is very significant, especially over the Rayleigh fading channel.

According to simulation result discussion of hard-decision list decoding given in section 6.7, using hard-decision decoding to achieve its optimal results remains almost infeasible. For example, to list decode Hermitian codes (64, 19, 40), (64, 29, 30), (64, 39, 20), (512, 153, 332) and (512, 289, 196) at the boundary τ_{GS} (3.32), the designed length of output list l = 28, 48, 13, 359 and 118 is required respectively. However, based on the performances of Figs 7.3 to 7.7, it can be observed that the soft-decision with small length of output list can outperform the hard-decision's optimal result, especially over the Rayleigh fading channel. For example, for the Hermitian code (512, 153, 332) over the Rayleigh fading channel, soft-decision list decoding with designed length $l = 2$ can outperform hard-decision decoding's optimal result. This phenomenon indicates that hard-decision's prohibitive optimal result can easily be achieved by using the soft-decision algorithm without very high decoding complexity.

7.6.2 Performance Assessment with a Large Length of Output List

According to the above analysis, a large length of output list demands high decoding complexity and it would be infeasible to achieve by simulation. In this thesis's analysis, $l \geq 5$ is considered to be a large length for the output list. However, the performance can still be assessed without applying the interpolation and factorisation processes. Assessments are made based on (7.22). After the reliability matrix $\boldsymbol{\Pi}$ is obtained, algorithm 7.1 is performed based on a designed length of output list l. Assuming the receiver acknowledges the transmitted code word $\bar{c}$ and based on the resulting multiplicity matrix $\boldsymbol{M}$, the code word score $S_M(\bar{c})$ can be determined by (7.15). Based on matrix $\boldsymbol{M}$, its cost C_M can be determined by (7.13) and so can the interpolated polynomial Q_M's $(1, w_z)$-weighted degree $\deg_{1,w_z}(Q_M(x, y, z))$ by (7.21). If $S_M(\bar{c}) > \Delta_{1,w_z}(C_M)$, then $S_M(\bar{c}) > \deg_{1,w_z}(Q_M(x, y, z))$. According to corollary 7.4, decoding is successful and the transmitted message polynomial *f* can be found out. Otherwise if $S_M(\bar{c}) \leq \Delta_{1,w_z}(C_M)$, decoding is a failure. From the performances presented in Figs 7.3 to 7.7, it can be observed that soft-decision list decoding achieves its performance improvement as the designed length of output list *l* increases. It should be noticed that these performance assessments are slightly worse compared to the practical results. Because based on (7.27), $\Delta_{1,w_z}(C_M)$ is the interpolated polynomials Q_M's (1, w_z)-weighted degree upper bound. Assessing the performance by (7.22), decoding is claimed to be failed if $S_M(\bar{c}) \leq \Delta_{1,w_z}(C_M)$. However, according to (7.27), $S_M(\bar{c})$ might still be greater than $\deg_{1,w_z}(Q_M(x, y, z))$. Based on corollary 7.4, it should result in a successful list decoding.

7.6.3 Asymptotically Optimal Performance Assessment

As soft-decision list decoding achieves its performance improvement by increasing the designed length of output list *l*, its optimal performance could be achieved when $l \to \infty$. In algorithm 7.1, if $l \to \infty$, $s \to \infty$ and multiplicity matrix ***M***'s entries $m_{i,j} \to \infty$. As a result, cost $C_M \to \infty$ as well as $\Delta_{1,w_z}(C_M) \to \infty$. Based on corollary 7.2, $\Delta_{1,w_z}(C_M) = \sqrt{2w_z N_{1,w_z}(\Delta_{1,w_z}(C_M))} = \sqrt{2w_z C_M}$. Therefore, when $l \to \infty$, successful list decoding assessment (7.22) can be written as:

$$S_M(\bar{c}) > \sqrt{2w_z C_M} \tag{7.30}$$

Based on (7.13) and (7.15), (7.30) could be equivalently written as:

$$\sum_{j=0}^{n-1}\{m_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} > \sqrt{w_z \sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}(m_{i,j}+1)} \tag{7.31}$$

According to lemma 5.4, when $s \to \infty$, $\frac{\pi_{i,j}}{n} \cong \frac{m_{i,j}}{s}$. Substituting $m_{i,j} = \frac{s}{n}\pi_{i,j}$ into (7.31) results in:

$$\frac{s}{n}\sum_{j=0}^{n-1}\{\pi_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} > \frac{s}{n}\sqrt{w_z \sum_{i=0}^{q-1}\sum_{j=0}^{n-1} \pi_{i,j}(\pi_{i,j}+\frac{n}{s})} \tag{7.32}$$

As $\frac{n}{s} \cong 0$ when $s \rightarrow \infty$, (7.32) can be approximately simplified as:

$$\sum_{j=0}^{n-1} \{\pi_{i,j} \mid \rho_i = c_j, i = 0,1,\ldots,q-1\} > \sqrt{w_z \sum_{i=0}^{q-1} \sum_{j=0}^{n-1} \pi_{i,j}^{\ 2}} \qquad (7.33)$$

Therefore, the soft-decision's optimal performance can be assessed based on (7.33). From the above analysis, it can be seen that the soft-decision's optimal performance is decided as far as the reliability matrix ***Π*** is obtained. Practical performance degradation is due to the designed length of output list constraint. Figs 7.3 to 7.7 show that the soft-decision approaches its optimal performance as the designed length of the output list increases. (7.33) also indicates that soft-decision list decoding has more potential improvements for low rate codes as they have lower w_z values. This is proven by the presented results. For example, over the AWGN channel, for Hermitian code (64, 19, 40), soft-decision decoding's optimal result has 1.4 dB coding gain at BER = 10^{-5} compared with hard-decision decoding's optimal result. For Hermitian code (64, 39, 20), soft-decision decoding's optimal result has about 1 dB coding gain.

7.7 Conclusion

Based on Koetter and Vardy's soft-decision scheme for Reed-Solomon codes, this chapter presented a soft-decision list decoding algorithm for one of the best performing algebraic-geometric codes – Hermitian codes. Different to the hard-decision list decoding algorithm, the received information's probabilistic reliability values are obtained by the receiver. The reliability values are then converted into multiplicity values based on which the interpolation is processed. During this conversion, a practical method for implementing the algorithm's stopping rule based on a designed length of the output list was suggested. It was shown that the soft-decision algorithm can produce a higher code word score for Hermitian codes than the hard-decision algorithm, and therefore increase the system's error-correction capability. For the soft-decision interpolation process, it was shown that the complexity could be reduced by identifying unnecessary polynomials and eliminating them during the iterations. Performances presented in this chapter showed that, based on the same length of output list, soft-decision list decoding has a significant coding gain over hard-decision but with higher decoding complexity. According to Chapter 6, achieving the hard-decision decoding optimal result remained infeasible because of high decoding complexity. However, this prohibitive result can be achieved by using the soft-decision list decoder with a short output list, especially over the Rayleigh fading channel. This soft-decision scheme can achieve further improvements by increasing the designed length of the output list. An asymptotic analysis of the soft-decision scheme showed that its optimal performance is decided by the probabilistic reliability value. Practical performance degradation is due to the length of output list constraint. This asymptotic analysis also showed that the soft-decision scheme has greater improvement for low rate codes.

Chapter 8

Conclusion and Future Work

8.1 Conclusion of the Thesis

The thesis presented an efficient list decoding system for Reed-Solomon and algebraic-geometric codes. The system was implemented using both hard-decision and soft-decision schemes. Under the hard-decision scheme, a received word is obtained and followed by the interpolation and factorisation processes in order to determine the list of most likely transmitted code words. The simulation results presented in the thesis showed that the hard-decision list decoding algorithm can outperform conventional unique decoding algorithms, namely the Berlekamp-Massey algorithm for Reed-Solomon codes and Sakata's algorithm with majority voting for Hermitian codes. This performance improvement is more significant for low rate codes. While under the soft-decision scheme, the received word's posterior transition probability information is obtained, which is then converted into multiplicity information and followed by the interpolation and factorisation processes as in hard-decision GS algorithm. It was shown that the soft-decision scheme can provide significant coding gains over the hard-decision scheme. It is more important to point out that by using the soft-decision scheme, the prohibitive optimal result of hard-decision list decoding can be achieved at a moderate decoding complexity. Therefore, the soft-decision list decoding scheme would be more suitable for decoding Reed-Solomon and Hermitian codes, instead of the GS algorithm.

To reduce the list decoding system's high complexity, an original modification scheme for the complexity dominant interpolation process was proposed in Chapter 4. As those polynomials with leading order greater than the total iteration number will not be chosen as the interpolated polynomial and have no information contributing to the chosen interpolated polynomial, they can be eliminated during the iterations. This scheme identifies and eliminates those polynomials during the iterations in order to avoid unnecessary calculations. This modification is a general scheme which can be later applied to both hard-decision and soft-decision list decoding systems for Reed-Solomon and algebraic-geometric codes provided the iteration number is known by the decoder. This modification scheme was adopted by the later chapters to improve the system's decoding efficiency.

A modified hard-decision list decoding scheme for algebraic-geometric codes was presented in Chapter 6, in which Hermitian codes are using as an algebraic-geometric coding scheme. Modification for list decoding of Hermitian codes is three-fold: First, a new algorithm to determine the corresponding coefficients between a Hermitian curve's pole basis monomials and zero basis functions is presented. This algorithm is performed a priori to the interpolation process. With the knowledge of these corresponding coefficients, the interpolation efficiency can be greatly improved. Second, the complexity reduction modification introduced in Chapter 4 was applied to the interpolation process. Third, a more general factorisation algorithm which can be applied to both Reed-Solomon and algebraic-geometric codes was presented from an implementation point of view. With these modifications, the author has achieved the first simulation results on list decoding of Hermitian codes which are defined in GF(16) and GF(64).

The thesis has also developed the first soft-decision list decoding algorithm for Hermitian codes, which is presented by Chapter 7. Similar to the soft-decision list decoding of Reed-Solomon codes, received word's posterior transition probabilities are obtained by the receiver rather than a hard-decision received word. These probabilities are then converted into multiplicity values based on which interpolation and factorisation processes for Hermitian codes are performed. It was shown that, based on the same designed length of output list, the soft-decision scheme has significant coding gains over the hard-decision scheme. By using the soft-decision scheme with small output length, the prohibitive optimal result of hard-decision list decoding can easily be achieved with only moderate decoding complexity.

8.2 Future Work

Although the list decoding system can provide improved decoding performance, it is at the expense of higher decoding complexity compared to the conventional unique decoding algorithms. When applying the list decoding system, there is a trade off between performance improvement and complexity. Firstly, the list decoding system has better performance than the unique decoding algorithms, but it is more complicated to implement. Secondly, for the list decoding system itself, performance can be improved by increasing the interpolation multiplicity or the length of the designed output list for the soft-decision scheme. However, decoding complexity also increases exponentially. Even though the complexity problem has been addressed in this thesis, the list decoding system's complexity is still high compared to the unique decoding algorithms. Therefore, in order to make the list decoding system more practical for industrial application, more work on reducing the system's decoding complexity should be investigated.

So far, only the mathematical framework for list decoding of Hermitian codes has been developed, while for other types of algebraic-geometric codes it is still unknown. Based on this thesis demonstration, it should not be difficult to develop a list decoding scheme for other algebraic-geometric codes provided the code's pole basis and zero basis can be defined. With the knowledge of these two bases functions, the zero condition of this code's interpolated polynomials can be defined so that valid interpolation and factorisation processes can be performed. The soft-decision list decoding scheme for other classes of algebraic-geometric codes can then be further developed.

Reference:

[1] V. D. Goppa, "Codes on algebraic curves," *Soviet math*, vol. Dok. 24, pp. 75-91, 1981.

[2] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Industrial Appl. Math*, vol. 8, pp. 300-304, 1960.

[3] E. R. Berlekamp, *Algebraic Coding Theory*. New York: McGraw Hill, 1968.

[4] J. L. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. 15, pp. 122-127, 1969.

[5] S. Sakata, J. Justesen, Y. Madelung, H. E. Jensen, and T. Høholdt, "Fast decoding of algebraic-geometric codes up to the designed minimum distance," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1672-1677, 1995.

[6] G. L. Feng and T. R. N. Rao, "Decoding algebraic-geometric codes up to the designed minimum distance," *IEEE Trans. Inform. Theory*, vol. 39, pp. 37-46, 1993.

[7] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1757-1767, 1999.

[8] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 49, pp. 2809-2825, 2003.

[9] T. Høholdt and R. R. Nielsen, "Decoding Hermitian codes with Sudan's algorithm," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (Lecture Notes in Computer Science)*, vol. 1719, H. I. N. Fossorier, S. Lin, and A. Pole, Ed. Berlin, Germany: Springer-Verlag, 1999, pp. 260-270.

[10] R. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Trans. Inform. Theory*, vol. 46, pp. 246-257, 2000.

[11] X.-W. Wu, "An algorithm for finding the roots of the polynomials over order domains," presented at ISIT 2002, Lausanne, Switzerland, 2002.

[12] X.-W. Wu and P. Siegel, "Efficient root-finding algorithm with application to list decoding of algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 2579-2587, 2001.

[13] O. Pretzel, *Codes and Algebraic Curves*. Oxford: Clarendon Press, 1998.

[14] M. A. Tsfasman, S. G. Vladut, and T. Zink, "Modular curves, Shimura curves and Goppa codes, better than Varshamov-Gilbert bound," *Math. Nachtrichten*, vol. 109, pp. 21-28, 1982.

[15] J. Justesen, K. J. Larsen, H. E. Jensen, A. Havemose, and T. Høholdt, "Construction and decoding of a class of algebraic geometry codes," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 811-821, 1989.

[16] G.-L. Feng and T. R. N. Rao, "A simple approach for construction of algebraic-geometric codes from affine plane curves," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1003-1012, 1994.

[17] C. Xing, H. Niederreiter, and K. Y. Lam, "Constructions of algebraic-geometry codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1186-1193, 1999.

[18] C. Heegard, J. Little, and K. Saints, "Systematic encoding via Grobner Bases for a class of algebraic-geometric Goppa codes," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1752-1761, 1995.

[19] I. Blake, C. Heegard, T. Høholdt, and V. Wei, "Algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 44, pp. 2596-2618, 1998.

[20] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equations for decoding Goppa codes," *Inform Contr*, vol. 27, pp. 87-99, 1975.

[21] D. M. Mandelbaum, "Decoding beyond the designed distance for certain algebraic codes," *Inform Contr*, vol. 29, pp. 207-228, 1977.

[22] W. W. Peterson, "Encoding and error-correction procedures for Bose-Chaudhuri codes," *IRE Transactions on Information Theory*, pp. 459-470, 1960.

[23] D. C. Gorenstein and N. Zierler, "A class of error-correcting codes in p^m symbols," *J. Soc. Industrial Appl. Math*, vol. 9, pp. 207-214, 1960.

[24] A. N. Skorobogatov and S. G. Vladut, "On decoding of algebraic geometric codes," *IEEE Trans. Inform. Theory*, vol. IT-36, pp. 1051-1060, 1990.

[25] S. Sakata, "Finding a minimal set of linear recurring relations capable of generating a given finite two-dimensional array," *J. Symbol. Comput*, vol. 5, pp. 321-337, 1998.

[26] S. Sakata, "Extension of the Berlekamp-Massey algorithm to *N* dimensions," *Inform. Computat*, vol. 84, pp. 207-239, 1990.

[27] J. Justesen, K. J. Larsen, H. E. Jensen, and T. Høholdt, "Fast decoding of codes from algebraic plane curves," *IEEE Trans. Inform. Theory*, vol. IT-38, pp. 1663-1676, 1992.

[28] C.-W. Liu, "Determination of error values for decodign Hermitian codes with inverser affine Fourier transform," *IEICE Trans. Fundamentals*, vol. E82-A, pp. 2302-2305, 1999.

[29] M. Johnston and R. A. Carrasco, "Construction and performance of algebraic-geometric codes over AWGN and fading channels," *IEE Proc Commun*, vol. 152, pp. 713-722, 2005.

[30] M. Johnston, R. A. Carrasco, and B. L. Burrows, "Design of new algebraic-geometric codes over fading channels," *IEE Electronic Letters*, 2004.

[31] M. Johnston, "Design and implementation of algebraic-geometric codes over AWGN and fading channels," in *School of electrical, electronic and computer engineering*. Newcastle-upon-Tyne: Newcastle University, 2006, pp. 190.

[32] G.-L. Feng and T. R. N. Rao, "Erasures-and-errors decoding of algebraic-geometric codes," presented at Proceeding of 1993 IEEE Inform. Theory workshop, 1993.

[33] S. Sakata, D. A. Leonard, H. E. Jensen, and T. Høholdt, "Fast erasure-and-error decoding of algebraic geometry codes up to teh Feng-Rao bound," *IEEE Trans. Inform. Theory*, vol. IT-44, pp. 1558-1564, 1998.

[34] M. Johnston and R. A. Carrasco, "Performance of Hermitian codes using combined error and erasure decoding," *IEE Proc Commun*, vol. 153, pp. 21 - 30, 2006.

[35] P. Elias, "LIst decoding for noisy channels," Res. Lab. Electron, MIT, Cambridge, MA 1957.

[36] P. Elias, "Error-correcting codes for list decoding," *Information Theory, IEEE Transactions on*, vol. 37, pp. 5-12, 1991.

[37] J. M. Wozencraft, "List decoding," Res. Lab. Electron, MIT, Cambridge, MA Jan 1958.

[38] S. Ar, R. Lipton, and M. Sudan, "Reconstructing algebraic functions from mixed data," *SIAM Journal on computing*, vol. 28, pp. 488-511, 1999.

[39] M. Sudan, "Decoding of Reed-Solomon codes beyond the error-correction bound," *J. Complexity*, vol. 13, pp. 180-193, 1997.

[40] M. Sudan, "Decoding of Reed-Solomon codes beyond the error-corrction diameter," presented at 35th Annual Allerton conference on communication, control and computing, 1997.

[41] M. Shokrollahi and H. Wasserman, "List decoding of algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 432-437, 1999.

[42] M. A. Shokrollahi and H. Wasserman, "Decoding algebraic-geometric codes beyond the error-correction bound," presented at 29th annual ACM symposium on theory of computing, 1998.

[43] V. Guruswami, *List decoding of error-correcting codes*. Berlin Heidelberg: Springer-Verlag, 2004.

[44] R. R. Nielsen, "List decoding of linear block codes." Lyngby, Denmark: Tech. Univ. Denmark, 2001.

[45] T. K. Moon, *Error correction coding - mathematical and algorithms*: Wiley Interscience, 2005.

[46] R. J. McEliece, "The Guruswami-Sudan decoding algorithm for Reed-Solomon codes," California Institute. Tech, Pasadena, California, IPN Progress Rep 42-153, 2003.

[47] G.-L. Feng and K. K. Tzeng, "A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with application to decoding cyclic codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1274-1287, 1991.

[48] S. H. Gao and M. A. Shokrollahi, "Computing roots of polynomials over function fields of curves," *preprint*, 1998.

[49] R. R. Nielsen, "A class of Sudan-decodable codes," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1564-1572, 2000.

[50] R. Koetter and A. Vardy, "A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes," presented at ITW-2003, Paris, France, 2003.

[51] T. Yaghoobian and I. F. Blake, "Reed-Solomon and Algebraic Geometry Codes," in *Reed-Solomon Codes and their Applications*, vol. Chapter 13, I. C. S. a. I. I. T. Society, Ed. Piscataway, NJ: IEEE Press, 1994, pp. 293-314.

[52] L. Chen, R. A. Carrasco, and E. G. Chester, "Performance of Reed-Solomon codes using the Guruswami-Sudan algorithm with improved interpolation efficiency," *IET Commun*, vol. 1, pp. 241 - 250, 2007.

[53] L. Chen, R. A. Carrasco, and E. G. Chester, "Decoding Reed-Solomon codes using the Guruswami-Sudan algorithm," presented at CSNDSP 2006, Patras Greece, 2006.

[54] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties and Algorithms*. New York: Springer-Verlag, 1992.

[55] G. D. Forney, "Generalised minimum distance decoding," *IEEE Trans. Inform. Theory*, vol. 12, pp. 125-131, 1966.

[56] H. Hasse, "Theorie der hoheren differentiale in einem algebraishen funcktionenkorper mit vollkommenem konstantenkorper nei beliebeger charakteristic," *J. Reine. Aug. Math*, vol. 175, pp. 50-54, 1936.

[57] W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*: Cambridge University Press, 2003.

[58] R. Koetter, "On algebraic decoding of algebraic-geometric and cyclic codes." Linkoping, Sweden: Univ. Linkoping, 1996.

[59] Y. Cassuto and J. Bruck, "On the average complexity of Reed-Solomon algebraic list decoder," presented at 8th International Symposium on Communication Theory and Applications (ISCTA), Ambleside, Lake district, UK, 2005.

[60] W. J. Gross, F. R. Kschischang, R. Koetter, and P. G. Gulak, "Application of algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Commun*, vol. 54, pp. 1224-1234, 2006.

[61] W. J. Gross, F. R. Kschischang, R. Koetter, and P. G. Gulak, "Towards a VLSI architecture for interpolation-based soft-decision Reed-Solomon decoders," *J. VLSI Signal Process*, vol. 39, pp. 93-111, 2005.

[62] E. R. Berlekamp, R. E. Peile, and S. P. Pope, "The application of error control to communications," *IEEE Commun. Mag*, vol. 25, pp. 44-57, 1987.

[63] A. Vardy and Y. Be'ery, "Bit-level soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 440-444, 1991.

[64] J. G. Proakis, *Digital Communications*, Fourth ed: McGraw-Hill International, 2000.

[65] L. Chen, R. A. Carrasco, and M. Johnston, "List decoding performance of algebraic geometric codes," *IET Electronic Letters*, vol. 42, 2006.

[66] L. Chen, R. A. Carrasco, and M. Johnston, "Reduced complexity interpolation for list decoding Hermitian codes," *IEEE Trans. Wireless Commmun*, Accepted for publication.

[67] L. Chen, R. A. Carrasco, M. Johnston, and E. G. Chester, "Efficient factorisation algorithm for list decoding algebraic-geometric and Reed-Solomon codes," presented at ICC 2007, Glasgow, UK, 2007.

[68] L. Chen and R. A. Carrasco, "Efficient list decoder for algebraic-geometric codes," presented at 9th International Symposium on Communication Theory and Application (ISCTA'07), Ambleside, Lake district, UK, 2007.

[69] L. Chen, R. A. Carrasco, and M. Johnston, "Soft-decision list decoding of Hermitian codes," *IEEE Trans. Commun*, Submitted for publication.

List of Symbols

Symbols	**Definitions**
d	The minimal Hamming distance
d^*	The designed minimal distance
k	Dimension of a code
n	Length of a code
r	Code rate / degree of an algebraic curve
κ	Relative minimum distance rate
R / $\overline{\Re}$	Received word (vector)
τ	Error-correction capability of a code
τ_{GS}	Error-correction capability of the Guruswami-Sudan algorithm
τ_{unique}	Error-correction capability of the unique decoding algorithm
GF(q)	Galois field with size q
w	Positive integers with any power of base number 2
χ	General algebraic curve
H_w	Hermitian curve defined in GF(w^2)
p_i	Affine point on an algebraic curve
p_∞	Point of infinity of an algebraic curve
N	Total number of affine points and point of infinity on an algebraic curve / Nonnegative integers
σ	Primitive element of a Galois field
v	Order (pole/zero) of a rational function
D, G	Divisors of an algebraic-geometric code
L	A sequence of rational functions
$l(G)$	Dimension of a code defined by devisor G
g	Genus of an algebraic curve
G_{RS}	Generator matrix for a Reed-Solomon code
G_{Herm}	Generator matrix for a Hermitian code
$f(x)$	Message polynomial
$\bar{c}$	Code word
ψ	Zero basis functions

ϕ	Pole basis functions
$Q(x, y)$ / $Q(x, y, z)$	Interpolated polynomial
m	Interpolation multiplicity
α, β	Zero parameters for interpolation
$F_q[x, y]$ / $F_q[x, y, z]$	A ring or polynomials with variables x, y, (z) and coefficients are chosen from Galois field GF(q)
$\deg_{1, k-1}(x^a y^b)$	(1, k - 1)-weighted degree of monomial $x^a y^b$
$\deg_{1,w_z}(\phi_a z^b)$	(1, w_z)-weighted degree of monomial $\phi_a z^b$
w_z	Weighted-degree of variable z
$S_x(T)$, $S_y(T)$	Parameters for hard-decision list decoding of Reed-Solomon codes
ord	Lexicographic order of a monomial ($x^a y^b$ / $\phi_a z^b$)
lod	Leading order of an interpolated polynomial ($Q(x, y)$ / $Q(x, y, z)$)
C_m / C_M	Total number of iterations in interpolation with respect to multiplicity m (hard-decision) / Total number of iterations in interpolation with respect to multiplicity matrix $\boldsymbol{M}$ (soft-decision)
τ_m	Error-correction capability of the list decoding algorithm by using interpolation multiplicity m
l_m	The maximal number of output list of the hard-decision list decoding algorithm using multiplicity m
l_M	The maximal number of output list of the soft decision list decoding algorithm using multiplicity matrix $\boldsymbol{M}$
i_k	Iteration index of the interpolation process
Δ	Hasse derivative evaluation of polynomial $Q(x, y)$ / zero condition evaluation of polynomial $Q(x, y, z)$
$D_{\alpha\beta}^{(x_i, y_i)}(Q)$	(α, β) Hasse derivative evaluation of polynomial $Q(x, y)$ at point (x_i, y_i)
$D_{\alpha\beta}^{(p_i, r_i)}(Q)$	(α, β) Hasse derivative evaluation of polynomial $Q(x, y, z)$ at point (p_i, r_i)
p / h	Candidate message polynomial from factorisation
$\Lambda(p, R)$ / $\Lambda(h, R)$	Number of code word symbols that the evaluation of p / h

	matches received word R
e	Error weighted of a hard-decision received word
$\gamma(e)$	Error dependent interpolation cost of an interpolated polynomial
$i_a(e)$	Error dependent index for the complexity reduction scheme
$N_{1,\,k-1}(\delta)$	The number of bivariate monomial $x^a y^b$ with (1, k-1)-weighted degree not greater than a nonnegative integer δ
$\Delta_{1,\,k-1}(v)$	The minimal value of δ that guarantees $N_{1,\,k-1}(\delta)$ is greater than a nonnegative integer v
$p(\cdot \mid \cdot)$	Probability density / mass function
$\Pr(\cdot \mid \cdot)$	Probability value
$\boldsymbol{\Pi}$	Reliability matrix
$\pi_{i,\,j}$	Entries of the reliability matrix
$\boldsymbol{M}$	Multiplicity matrix
$m_{i,\,j}$	Entries of the multiplicity matrix
S	Modulated symbols
N_0	Power of noise
s	A desire value for initialization of Algorithm 5.1 / Algorithm 7.1
$S_M(\bar{c})$	Code word score with respect to multiplicity matrix $\boldsymbol{M}$
$\gamma_{a,p_i,\alpha}$	The corresponding coefficient between a Hermitian curve's pole basis function ϕ_a and zero basis function $\psi_{p_i,\alpha}$
t_m	Hard-decision list decoding parameter for Hermitian code
V	Classes of interpolated polynomial for Algorithm 6.2
δ, u, λ	Nonnegative integers for defining classes V
$\phi_L^{(s)}$ / $C_L^{(s)}$	Leading monomial / leading coefficient of polynomial $Q^{(s)}$ during the factorisation process for Hermitian codes
$N_{1,w_z}(\delta)$	The number of monomials with (1, w_z)-weighted degree not greater than δ
$\Delta_{1,w_z}(v)$	The minimal value of δ that guarantees $N_{1,w_z}(\delta)$ is greater v

List of Abbreviations

Abbreviations	**Definitions**
AG	Algebraic-geometric codes
AWGN	Additive white Gaussian noise
BM	Berlekamp-Massey algorithm
BPSK	Binary phase shift keying
CD	Compact disc
DVD	Digital versatile disc
GS	Guruswami-Sudan algorithm
GMD	Generalised minimum distance
KV	Koetter-Vardy algorithm
LC	Leading coefficient
LM	Leading monomial
QPSK	Quadrature phase shift keying
RS	Reed-Solomon codes
RCS	Recursive coefficient search

List of Figures

List of Tables

Appendix A: Finite Field Calculation of GF(4)

σ is a primitive element in GF(4) satisfying $\sigma^2 + \sigma + 1 = 0$.

Addition Table:

$+$	**0**	**1**	$\boldsymbol{\sigma}$	$\boldsymbol{\sigma^2}$
0	0	1	σ	σ^2
1	1	0	σ^2	σ
$\boldsymbol{\sigma}$	σ	σ^2	0	1
$\boldsymbol{\sigma^2}$	σ^2	σ	1	0

Multiplication Table:

$\times$	**0**	**1**	$\boldsymbol{\sigma}$	$\boldsymbol{\sigma^2}$
0	0	0	0	0
1	0	1	σ	σ^2
$\boldsymbol{\sigma}$	0	σ	σ^2	1
$\boldsymbol{\sigma^2}$	0	σ^2	1	σ

Appendix B: Finite Field Calculation of GF(8)

σ is a primitive element in GF(8) satisfying $\sigma^3 + \sigma + 1 = 0$.

Addition Table:

+	**0**	**1**	$\boldsymbol{\sigma}$	$\boldsymbol{\sigma^3}$	$\boldsymbol{\sigma^2}$	$\boldsymbol{\sigma^6}$	$\boldsymbol{\sigma^4}$	$\boldsymbol{\sigma^5}$
0	0	1	σ	σ^3	σ^2	σ^6	σ^4	σ^5
1	1	0	σ^3	σ	σ^6	σ^2	σ^5	σ^4
$\boldsymbol{\sigma}$	σ	σ^3	0	1	σ^4	σ^5	σ^2	σ^6
$\boldsymbol{\sigma^3}$	σ^3	σ	1	0	σ^5	σ^4	σ^6	σ^2
$\boldsymbol{\sigma^2}$	σ^2	σ^6	σ^4	σ^5	0	1	σ	σ^3
$\boldsymbol{\sigma^6}$	σ^6	σ^2	σ^5	σ^4	1	0	σ^3	σ
$\boldsymbol{\sigma^4}$	σ^4	σ^5	σ^2	σ^6	σ	σ^3	0	1
$\boldsymbol{\sigma^5}$	σ^5	σ^4	σ^6	σ^2	σ^3	σ	1	0

Multiplication Table:

×	**0**	**1**	$\boldsymbol{\sigma}$	$\boldsymbol{\sigma^3}$	$\boldsymbol{\sigma^2}$	$\boldsymbol{\sigma^6}$	$\boldsymbol{\sigma^4}$	$\boldsymbol{\sigma^5}$
0	0	0	0	0	0	0	0	0
1	0	1	σ	σ^3	σ^2	σ^6	σ^4	σ^5
$\boldsymbol{\sigma}$	0	σ	σ^2	σ^4	σ^3	1	σ^5	σ^6
$\boldsymbol{\sigma^3}$	0	σ^3	σ^4	σ^7	σ^5	σ^2	1	σ
$\boldsymbol{\sigma^2}$	0	σ^2	σ^3	σ^5	σ^4	σ	σ^6	1
$\boldsymbol{\sigma^6}$	0	σ^6	1	σ^2	σ	σ^5	σ^3	σ^4
$\boldsymbol{\sigma^4}$	0	σ^4	σ^5	1	σ^6	σ^3	σ	σ^2
$\boldsymbol{\sigma^5}$	0	σ^5	σ^6	σ	1	σ^4	σ^2	σ^3

Appendix C: Finite Field Calculation of GF(16)

σ is a primitive element in GF(16) satisfying $\sigma^4 + \sigma + 1 = 0$.

Addition Table:

+	**0**	**1**	$\boldsymbol{\sigma}$	$\boldsymbol{\sigma^4}$	$\boldsymbol{\sigma^2}$	$\boldsymbol{\sigma^8}$	$\boldsymbol{\sigma^5}$	$\boldsymbol{\sigma^{10}}$	$\boldsymbol{\sigma^3}$	$\boldsymbol{\sigma^{14}}$	$\boldsymbol{\sigma^9}$	$\boldsymbol{\sigma^7}$	$\boldsymbol{\sigma^6}$	$\boldsymbol{\sigma^{13}}$	$\boldsymbol{\sigma^{11}}$	$\boldsymbol{\sigma^{12}}$
0	0	1	σ	σ^4	σ^2	σ^8	σ^5	σ^{10}	σ^3	σ^{14}	σ^9	σ^7	σ^6	σ^{13}	σ^{11}	σ^{12}
1	1	0	σ^4	σ	σ^8	σ^2	σ^{10}	σ^5	σ^{14}	σ^3	σ^7	σ^9	σ^{13}	σ^6	σ^{12}	σ^{11}
$\boldsymbol{\sigma}$	σ	σ^4	0	1	σ^5	σ^{10}	σ^2	σ^8	σ^9	σ^7	σ^3	σ^{14}	σ^{11}	σ^{12}	σ^6	σ^{13}
$\boldsymbol{\sigma^4}$	σ^4	σ	1	0	σ^{10}	σ^5	σ^8	σ^2	σ^7	σ^9	σ^{14}	σ^3	σ^{12}	σ^{11}	σ^{13}	σ^6
$\boldsymbol{\sigma^2}$	σ^2	σ^8	σ^5	σ^{10}	0	1	σ	σ^4	σ^6	σ^{13}	σ^{11}	σ^{12}	σ^3	σ^{14}	σ^9	σ^7
$\boldsymbol{\sigma^8}$	σ^8	σ^2	σ^{10}	σ^5	1	0	σ^4	σ	σ^{13}	σ^6	σ^{12}	σ^{11}	σ^{14}	σ^3	σ^7	σ^9
$\boldsymbol{\sigma^5}$	σ^5	σ^{10}	σ^2	σ^8	σ	σ^4	0	1	σ^{11}	σ^{12}	σ^6	σ^{13}	σ^9	σ^7	σ^3	σ^{14}
$\boldsymbol{\sigma^{10}}$	σ^{10}	σ^5	σ^8	σ^2	σ^4	σ	1	0	σ^{12}	σ^{11}	σ^{13}	σ^6	σ^7	σ^9	σ^{14}	σ^3
$\boldsymbol{\sigma^3}$	σ^3	σ^{14}	σ^9	σ^7	σ^6	σ^{13}	σ^{11}	σ^{12}	0	1	σ	σ^3	σ^2	σ^8	σ^5	σ^{10}
$\boldsymbol{\sigma^{14}}$	σ^{14}	σ^3	σ^7	σ^9	σ^{13}	σ^6	σ^{12}	σ^{11}	1	0	σ^4	σ	σ^8	σ^2	σ^{10}	σ^5
$\boldsymbol{\sigma^9}$	σ^9	σ^7	σ^3	σ^{14}	σ^{11}	σ^{12}	σ^6	σ^{13}	σ	σ^4	0	1	σ^5	σ^{10}	σ^2	σ^8
$\boldsymbol{\sigma^7}$	σ^7	σ^9	σ^{14}	σ^3	σ^{12}	σ^{11}	σ^{13}	σ^6	σ^4	σ	1	0	σ^{10}	σ^5	σ^8	σ^2
$\boldsymbol{\sigma^6}$	σ^6	σ^{13}	σ^{11}	σ^{12}	σ^3	σ^{14}	σ^9	σ^7	σ^2	σ^8	σ^5	σ^{10}	0	1	σ	σ^4
$\boldsymbol{\sigma^{13}}$	σ^{13}	σ^6	σ^{12}	σ^{11}	σ^{14}	σ^3	σ^7	σ^9	σ^8	σ^2	σ^{10}	σ^5	1	0	σ^4	σ
$\boldsymbol{\sigma^{11}}$	σ^{11}	σ^{12}	σ^6	σ^{13}	σ^9	σ^7	σ^3	σ^{14}	σ^5	σ^{10}	σ^2	σ^8	σ	σ^4	0	1
$\boldsymbol{\sigma^{12}}$	σ^{12}	σ^{11}	σ^{13}	σ^6	σ^7	σ^9	σ^{14}	σ^3	σ^{10}	σ^5	σ^8	σ^2	σ^4	σ	1	0

Multiplication Table:

$\times$	**0**	**1**	σ	σ^4	σ^2	σ^8	σ^5	σ^{10}	σ^3	σ^{14}	σ^9	σ^7	σ^6	σ^{13}	σ^{11}	σ^{12}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	σ	σ^4	σ^2	σ^8	σ^5	σ^{10}	σ^3	σ^{14}	σ^9	σ^7	σ^6	σ^{13}	σ^{11}	σ^{12}
σ	0	σ	σ^2	σ^5	σ^3	σ^9	σ^6	σ^{11}	σ^4	1	σ^{10}	σ^8	σ^7	σ^{14}	σ^{12}	σ^{13}
σ^4	0	σ^4	σ^5	σ^8	σ^6	σ^{12}	σ^9	σ^{14}	σ^7	σ^3	σ^{13}	σ^{11}	σ^{10}	σ^2	1	σ
σ^2	0	σ^2	σ^3	σ^6	σ^4	σ^{10}	σ^7	σ^{12}	σ^5	σ	σ^{11}	σ^9	σ^8	1	σ^{13}	σ^{14}
σ^8	0	σ^8	σ^9	σ^{12}	σ^{10}	σ	σ^{13}	σ^3	σ^{11}	σ^7	σ^2	1	σ^{14}	σ^6	σ^4	σ^5
σ^5	0	σ^5	σ^6	σ^9	σ^7	σ^{13}	σ^{10}	1	σ^8	σ^4	σ^{14}	σ^{12}	σ^{11}	σ^3	σ	σ^2
σ^{10}	0	σ^{10}	σ^{11}	σ^{14}	σ^{12}	σ^3	1	σ^5	σ^{13}	σ^9	σ^4	σ^2	σ	σ^8	σ^6	σ^7
σ^3	0	σ^3	σ^4	σ^7	σ^5	σ^{11}	σ^8	σ^{13}	σ^6	σ^2	σ^{12}	σ^{10}	σ^9	σ	σ^{14}	1
σ^{14}	0	σ^{14}	1	σ^3	σ	σ^7	σ^4	σ^9	σ^2	σ^{13}	σ^8	σ^6	σ^5	σ^{12}	σ^{10}	σ^{11}
σ^9	0	σ^9	σ^{10}	σ^{13}	σ^{11}	σ^2	σ^{14}	σ^4	σ^{12}	σ^8	σ^3	σ	1	σ^7	σ^5	σ^6
σ^7	0	σ^7	σ^8	σ^{11}	σ^9	1	σ^{12}	σ^2	σ^{10}	σ^6	σ	σ^{14}	σ^{13}	σ^5	σ^3	σ^4
σ^6	0	σ^6	σ^7	σ^{10}	σ^8	σ^{14}	σ^{11}	σ	σ^9	σ^5	1	σ^{13}	σ^{12}	σ^4	σ^2	σ^3
σ^{13}	0	σ^{13}	σ^{14}	σ^2	1	σ^6	σ^3	σ^8	σ	σ^{12}	σ^7	σ^5	σ^4	σ^{11}	σ^9	σ^{10}
σ^{11}	0	σ^{11}	σ^{12}	1	σ^{13}	σ^4	σ	σ^6	σ^{14}	σ^{10}	σ^5	σ^3	σ^2	σ^9	σ^7	σ^8
σ^{12}	0	σ^{12}	σ^{13}	σ	σ^{14}	σ^5	σ^2	σ^7	1	σ^{11}	σ^6	σ^4	σ^3	σ^{10}	σ^8	σ^9

Appendix D: Hard-Decision List Decoding Parameters of Some Hermitian Codes

Hermitian code (64, 19, 40):

$v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{18}{}^{-1}) = v_{p_\infty}((xy^4)^{-1}) = 24$, $\tau_{GS} = 24$, and

m	1	2	3	4	5	8	17
l_m	2	3	5	7	8	13	28
t_m	2	18	9	1	16	14	7
τ_m	13	18	20	21	22	23	$24 = \tau_{GS}$

Hermitian code (64, 29, 30):

$v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{28}{}^{-1}) = v_{p_\infty}((xy^6)^{-1}) = 34$, $\tau_{GS} = 17$, and

m	1	2	3	4	5	9	35
l_m	1	3	4	5	7	12	48
t_m	21	2	14	27	6	23	12
τ_m	8	11	13	14	15	16	$17 = \tau_{GS}$

Hermitian code (64, 39, 20):

$v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{38}{}^{-1}) = v_{p_\infty}((xy^8)^{-1}) = 44$, $\tau_{GS} = 10$, and

m	1	2	3	4	6	11
l_m	1	2	3	5	7	13
t_m	16	26	36	2	20	21
τ_m	3	6	7	8	9	$10 = \tau_{GS}$

Hermitian code (512, 153, 332):

$v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{152}^{-1}) = v_{p_\infty}((y^{20})^{-1}) = 180$, $\tau_{GS} = 208$, and

m	1	2	3	4	5	6	7	8	9	10	11	12	14	17	20	26	37	63	213
l_m	2	3	5	7	8	10	12	13	15	17	19	20	24	29	34	44	62	106	359
t_m	13	142	90	37	161	105	50	174	118	62	6	129	17	28	39	61	161	134	131
τ_m	138	170	181	187	191	194	196	197	198	199	200	201	202	203	204	205	206	207	208 = τ_{GS}

Hermitian code (512, 289, 196):

$v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{288}^{-1}) = v_{p_\infty}((x^8y^{28})^{-1}) = 316$, $\tau_{GS} = 109$, and

m	1	2	3	4	5	6	7	8	9	11	12	15	19	26	40	93
l_m	1	2	4	5	6	7	9	10	11	14	15	19	24	33	51	118
t_m	126	224	6	91	177	266	39	123	210	68	154	98	127	99	43	190
τ_m	69	83	88	94	97	98	100	101	102	103	104	105	106	107	108	109 = τ_{GS}

Note: multiplicity m listed in the above tables are the minimal value in order to have the corresponding error-correction capability lower bound τ_m for hard-decision list decoding of Hermitian codes.